

```
<style>
  body,div{
    margin:0;
  }
  html{
    margin-top:50px;
  }
  div{
    width:100px;
    height:100px;
    margin-top:50px;
    background-color:red;
  }
</style>
<div></div>
```

CSS



核心技术详解

肖志华 著

电子工业出版社

Publishing House of Electronics Industry

北京•BEIJING

内 容 简 介

本书共分 13 章，第 1 章主要解答 CSS 中常见的问题，以及常用的技巧。第 2~6 章讲解了 CSS 的核心技术，其中第 2 章是最为核心的内容，相对于其他章节理解起来会比较难一点。第 3~6 章主要介绍案例，配合第 2 章阅读会轻松很多。本书每个章节都是独立的，因此如果某些章节看不懂，可以暂且跳过，先阅读其他章节。第 7~13 章讲解关于 CSS 3 的内容。

本书内容精练、实例丰富、通俗易懂，可作为广大 CSS 设计人员和前端开发人员的参考书，同时也非常适合大中专院校师生学习阅读。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

CSS 核心技术详解 / 肖志华著. —北京：电子工业出版社，2017.6

ISBN 978-7-121-31330-1

I. ①C… II. ①肖… III. ①网页制作工具 IV.①TP393.092.2

中国版本图书馆 CIP 数据核字（2017）第 076255 号

策划编辑：黄爱萍

责任编辑：徐津平

印 刷：北京中新伟业印刷有限公司

装 订：北京中新伟业印刷有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱

邮编：100036

开 本：787×980 1/16 印张：20.5 字数：409 千字

版 次：2017 年 6 月第 1 版

印 次：2017 年 6 月第 1 次印刷

定 价：59.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlt@s@phei.com.cn，盗版侵权举报请发邮件至 dbqq@s@phei.com.cn。

本书咨询联系方式：（010）51260888-819，faq@s@phei.com.cn。

前言

看似简单的 CSS，却暗藏玄机，那是我们摸爬滚打好长时间后悟出的真理。

在很长的一段时间里，我并没有重视 CSS，觉得 CSS 很简单，无非就是一些属性；后来才发现自己小看了 CSS，对 CSS 的了解实在是太少，尤其是对其核心概念的理解太模糊，实际上它有很多神奇的地方并不为大家所知。对于一个新手来说，只知道一些理论但在实际开发中不会使用是不行的，于是笔者萌生了写作本书的最初想法。

市面上介绍 CSS 基础的书已经有很多了，已经没有必要再去重复，但是一些核心的内容还是很有必要写出来的，因为我发现很多前端朋友对 CSS 都不太重视。我认为做前端的技术人员不仅要掌握好 CSS 的核心内容，还要懂得怎样把这些内容灵活运用 to 实际开发中。如果对一门技术只停留在了了解的层面而不会使用，那和不会有什么区别？所以本书将实用放在第一位，大量的例子都来自于我在实际开发中所遇到的问题，将这些实际的例子拿来讲解才更有说服力，同时也更易于读者的理解。

本书共分 13 章，第 1 章主要解答 CSS 中常见的问题，以及常用的技巧。第 2~6 章讲解了 CSS 的核心技术，其中第 2 章是最为核心的内容，相对于其他章节理解起来会比较难一点。第 3~6 章主要介绍案例，每个代码片段都有一些案例，配合第 2 章阅读会轻松很多。本书每个章节都是独立的，因此如果某些章节看不懂，可以暂且跳过，先阅读其他章节。第 7~13 章讲解关于 CSS 3 的内容。

本书举例时用到了很多关于 CSS 3 的属性，所以读者在测试时需要使用高级浏览器，这里推荐使用 Chrome 浏览器，书中的例子主要也是在 Chrome 浏览器中测试的。另外本书并不会过多地讲解兼容性问题，因为花太多时间讨论兼容性是不太值得的。书中有一些个人看法，由于才疏学浅，不免会有疏漏。如果发现错误，还请指出，不吝赐教，在此深表

谢意，可发邮件至 c776@foxmail.com 邮箱，一定一一回复并乐此不疲，因为这是我的工作，和你们交流也是我的快乐。

本书的出版要特别感谢电子工业出版社的黄爱萍和张童编辑，感谢他们在选题策划和书稿编辑方面做出的大量工作，同时对伯乐在线黄利民大哥的大力支持深表谢意。

肖志华

2017 年 3 月 28 日

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **提交勘误：**您对书中内容的修改意见可在【提交勘误】处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **与作者交流：**在页面下方【读者评论】处留下您的疑问或观点，与作者和其他读者一同学习交流。

页面入口：<http://www.broadview.com.cn/31330>



目 录 ➞

第 1 章 遇见未知的 CSS / 1

- 1.1 在 CSS 中会遇到的问题 / 1
 - 1.1.1 CSS 层叠规则 / 3
 - 1.1.2 CSS 的命名 / 5
- 1.2 CSS 的一些技巧 / 6
 - 1.2.1 使用 pointer-events 控制鼠标事件 / 6
 - 1.2.2 玩转 CSS 选择器 / 8
 - 1.2.3 利用 padding 实现元素等比例缩放 / 11
 - 1.2.4 calc 函数 / 14
- 1.3 隐藏元素 / 18

第 2 章 CSS 核心概念 / 23

- 2.1 CSS 解析规则 / 23
- 2.2 替换元素与非替换元素 / 28
- 2.3 属性值的计算规则 / 28
- 2.4 可视化格式模型 / 30
- 2.5 包含块 / 31
- 2.6 控制框 / 38
- 2.7 格式化上下文 BFC、IFC / 40
 - 2.7.1 从 overflow 清除浮动看 BFC（块格式化上下文） / 40

2.7.2 块级格式化上下文 BFC / 45

2.7.3 折叠外边距 / 54

2.7.4 行内格式化上下文 IFC / 58

2.7.5 行高的计算 / 61

第 3 章 CSS 单位究竟来自何方 / 67

3.1 百分比究竟为谁 / 67

3.2 探索 auto 密码 / 82

3.3 设计响应式网页 rem / 93

3.4 vw、vh、vmin、vmax 基于视口单位 / 97

3.5 什么是 ch / 102

3.6 min、max 的巧妙运用 / 104

3.7 一个 none 引出的大学问 / 106

第 4 章 那些年我们一起定位过的元素 / 108

4.1 定位的特点 / 108

4.1.1 定位之 absolute 篇 / 109

4.1.2 定位之 relative 篇 / 113

4.1.3 当定位遇到定位 / 117

4.1.4 定位之 fixed 篇 / 121

4.1.5 偶遇定位 bug，才知定位的真理 / 122

4.1.6 定位之 static 篇 / 129

4.2 透彻研究定位隐藏的秘密 / 130

4.3 总结 / 140

第 5 章 元素的七十二变——元素转换 / 142

5.1 display 介绍 / 142

5.2 大块头——block / 142

5.3 我们一起站一排——inline / 143

5.4 inline 和 block 的结合体——inline-block / 149

5.5 行内和块的烦恼 / 152

5.6 display 的一些其他属性 / 155

5.7 总结 / 159

第 6 章 浮动趣事 / 160

6.1 浮动简介 / 160

6.2 浮动的特点 / 161

6.3 浮动的秘密 / 167

6.4 实现任意形状的文字环绕 / 173

6.5 总结 / 188

第 7 章 再不学这些选择器就老了 / 189

7.1 那些被遗忘的选择器 / 189

7.1.1 相邻兄弟选择器 / 189

7.1.2 利用 hover 实现一个下拉菜单 / 192

7.1.3 利用 active 做一个集能量 / 194

7.1.4 用 first-letter 选中第一个字 / 195

7.1.5 用 first-line 选择首行文字 / 197

7.2 模拟父级选择器 / 199

7.3 强大的新选择器 / 200

第 8 章 CSS 图标制作 / 210

8.1 隐藏在边框中的秘密 / 210

8.2 边框的烦恼 / 212

8.3 边框的孪生兄弟——outline / 215

8.4 纯 CSS 图标制作 / 220

第 9 章 你今天换背景了吗 / 232

9.1 对背景属性的深入探索 / 232

9.2 新增的背景功能 / 237

9.2.1 改变背景原点——background-origin / 237

9.2.2 背景裁剪——background-clip / 239

9.2.3 设置背景图片大小——background-size / 243

9.3 总结 / 245

第 10 章 让文字更美一些 / 246

10.1 制作非主流文字 / 247

10.2 新增的文字对齐属性 / 250

10.2.1 文字两端对齐 / 250

10.2.2 末尾文本对齐 / 252

10.2.3 文本书写模式 / 257

10.3 关于文字的一些其他属性 / 259

10.3.1 将超出宽度的文字隐藏 / 259

10.3.2 字母转换大小写 / 262

10.4 总结 / 263

第 11 章 内容生成技术——用 CSS 来计数 / 264

11.1 伪元素 / 264

11.2 CSS 计数器 / 265

11.3 content 的其他用途 / 272

11.4 总结 / 273

第 12 章 解决让人头疼的布局 / 274

12.1 制作可自适应的布局 / 274

12.1.1 左侧固定、右侧自适应的布局 / 274

12.1.2 右侧固定、左侧自适应的布局 / 276

12.1.3	多列文字垂直居中	/	278
12.2	利用伸缩盒模型进行布局	/	283
12.2.1	伸缩盒模型基础	/	285
12.2.2	伸缩盒模型进阶	/	296
12.2.3	伸缩盒模型实战	/	299
第 13 章 飞越 CSS / 303			
13.1	CSS 最佳实践	/	303
13.2	纯 CSS 的世界	/	307
13.2.1	利用 checked 选择器实现 tab 切换	/	308
13.2.2	利用:target 选择器实现遮罩层效果	/	310
13.2.3	scaleY 配合 animation 制作 loading	/	311
13.2.4	利用 hover 实现手风琴效果	/	313
13.2.5	利用 checked 选择器制作星星评分	/	314
13.2.6	使用 flex 伸缩盒模型实现瀑布流布局	/	316
13.3	结束语	/	318



第 1 章 遇见未知的 CSS

1.1 在 CSS 中会遇到的问题

1. 在 CSS 中为什么要有层叠

在 CSS 中可能会有多个样式表同时影响同一个元素的某个属性，设计这个功能的主要原因有两个，解决模块化和作者、用户、用户代理样式冲突。

(1) 模块化

一个页面中的样式可以拆分成多个样式表，代码如下。

```
@import url(style/base.css);  
@import url(style/layer.css);
```

但这种方式也会随之产生一个问题，即如果对某个元素的同一个属性设置样式，到底应用谁的呢？

(2) 作者/用户/用户代理

当作者（写代码的人）和用户（浏览页面的人），以及用户代理（一般指浏览器）都能更改样式表时，也会产生同样的问题：究竟用谁设置的样式，因此 CSS 层叠机制就显得格外重要。

2. 为什么“@import”指令需要写在样式表的开头

代码如下。

```
@import url(style/layer.css);  
body{  
    background-color:red;  
}
```

“@import”指令之所以需要写在样式表的开头，是因为可以使后面的样式能更好地层叠导入进来的样式。

3. 当 CSS 值为 0 时为什么可以省略单位

因为当 CSS 值为 0 时，任何单位的结果都是一样的，就像数学中的 0 乘以任何数都得 0。

4. margin 垂直外边距折叠的意义是什么

margin 垂直外边距折叠的特性主要来自传统排版，举个例子，代码如下。

```
<style>
  body,ul,li{
    margin:0;
    padding:0;
  }
  ul{
    list-style:none;
  }
  ul>li{
    margin:20px 0;
  }
</style>
<ul>
  <li>1111111111</li>
  <li>2222222222</li>
  <li>3333333333</li>
</ul>
```

代码的效果如图 1.1 所示。

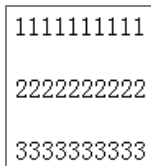


图 1.1 margin 外边距折叠效果

从图 1.1 中可以看到 3 行数字的垂直外边距都是一样的。如果没有这个特性，第一行数字与下面两行数字的外边距就不一样了，因为我们给每个 li 都设置了一个上下外边距，假如没有外边距折叠，那么第 1 个 li 的下边距加上第 2 个 li 的上边距，就是两倍的间距了，但是第一个 li 上面没有其他元素，所以它只有一个上边距，最终导致的结果就是，第 1 个 li 和后面的几个 li 的外边距不一样，这显然不是我们所希望的。而 margin 外边距折叠功能正是要在这种情况下，让格式可以好看一点。

1.1.1 CSS 层叠规则

在介绍 CSS 层叠规则之前首先举个例子，代码如下。

```
<style>
  .box{
    color:red;
    font-size:18px;
  }
</style>
<div class="box">
  <a href="#">层叠</a>
</div>
```

代码的效果如图 1.2 所示。

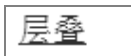


图 1.2 继承的颜色被层叠

按理说颜色是可以继承的，那么为什么 a 标签的颜色没有变成红色呢？审查一下元素，如图 1.3 所示。

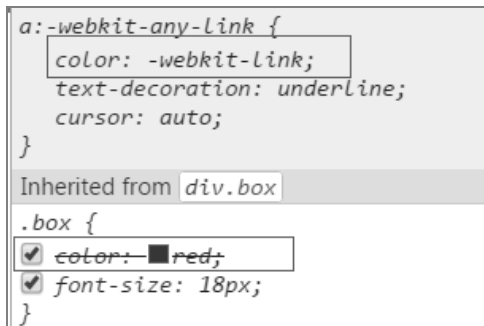


图 1.3 Chrome 浏览器中的效果

从图 1.3 中可以看到继承的颜色被划掉了，出现这个问题的原因是浏览器对 a 标签设置了默认样式，将继承的样式层叠了，因为继承的样式权重最小。下面介绍 CSS 关于层叠规则是怎么计算的。

在 CSS 中一个样式可能会来自不同的地方，分别是作者、用户及用户代理。如果在这几个样式中，它们对同一个元素的同一个属性做了不同的操作，那么用户代理应该如何处理这段 CSS 呢？举个例子，代码如下。

```
/* 作者 */
.box{
  color:red;
}
/* 用户代理 */
.box{
  color:green;
}
/* 用户 */
.box{
  color:pink;
}
```

可以看到用户代理、用户和作者的代码起冲突了，而 CSS 的层叠规则就是为了解决这些冲突，以下是一些 CSS 层叠规则。

在层叠中每个样式规则都有一个权重值，当其中几条规则同时生效时，权重值最大的规则优先。一般来说作者指定的样式权重值高于用户样式权重值，用户样式权重值高于客户端（用户代理）权重值。

在层叠顺序中，以下权重值从小到大。

- (1) 用户代理样式；
- (2) 用户一般样式；
- (3) 作者一般样式；
- (4) 作者重要样式 (!important)；
- (5) 用户重要样式 (!important)；

(6) 如果是两个样式来自相同的代码，如都来自作者（代码），并且它们的样式声明同样重要，则根据特异度来计算，特异度高的会覆盖特异度低的；

(7) 如果特异度也相同，则越往后的样式优先级越高。

1. !important 声明规则

!important 声明的样式比一般声明优先级高，并且用户设置的 !important 比作者设置的 !important 优先级高。这样做的原因是为了方便用户实现一些特殊的要求，例如页面字体大小的调整等。

下面举一个 !important 规则的例子，代码如下。

```
<style>
  .box{
    color:red !important;
```

```

    }
    .box{
        color:green;
    }
</style>
<div class="box">!important</div>

```

在正常情况下,后一个“color:green”会层叠前一个“color:red”,但这里我们给“color:red”设置了!important规则,所以前一个优先级高。

2. 选择器特异度的计算

- (1) 如果一个声明出现在元素的 style 属性中,则将 a 计为 1;
- (2) b 等于选择器中所有 id 选择器加起来的数量和;
- (3) c 等于选择器中所有 class 选择器和属性选择器,以及伪类选择器加起来的数量和;
- (4) d 等于选择器中所有标签选择器和伪元素选择器加起来的数量和。

将 a、b、c、d 这 4 个数字连接起来(a-b-c-d)就组成了选择器的特异度。一段特异度的计算,如下所示。

```

.box{                /* a=0 b=0 c=1 d=0 特异度 = 0,0,1,0 */
.box div{           /* a=0 b=0 c=1 d=1 特异度 = 0,0,1,1 */
#nav li{            /* a=0 b=1 c=0 d=1 特异度 = 0,1,0,1 */
p:first-line{       /* a=0 b=0 c=0 d=2 特异度 = 0,0,0,2 */
style="             /* a=1 b=0 c=0 d=0 特异度 = 1,0,0,0 */

```

它们的比较顺序是先比较 a,如果 a 的值都相同,那么接着比较 b、c、d 的值,谁的数大则优先级就越高。

在使用 CSS 选择器时,你需要注意以下两点。

- 继承的优先级最低,没有特异度;
- 结合符(如+、>等)及通用选择符(*)特异度为 0。

因此,可以知道之前 a 标签 color 属性为什么没有被应用了,因为继承的优先级最低。

1.1.2 CSS 的命名

在代码复用时,也许你写过类似以下这样的代码,代码如下。

```

size-10{
    font-size:10px;
}

```

虽然这段代码看起来没什么问题，但如果考虑到可维护性，那就有很大问题了。假如有一天你不想用 10px，想改成 12px，那么直接再加一个 class 就行了，修改后的代码如下。

```
size-10{  
  font-size:10px;  
}  
size-12{  
  font-size:12px;  
}
```

但那些页面中原本用的 size-10 的 class 都得修改成 size-12，所以不建议这么修改代码。笔者建议用语义的方式命名，代码如下。

```
.size-small{  
  font-size:12px;  
}
```

这样写代码的好处是当需要调整字体大小时，只需修改一处，而不必修改添加到元素上的 class。也就是说不要按照显示的效果命名，而是根据这个 class 的用意命名。

1.2 CSS 的一些技巧

1.2.1 使用 pointer-events 控制鼠标事件

可以用 CSS 中的 pointer-events 来控制元素什么时候响应鼠标事件，比较常用的一个场景是获取验证码，如图 1.4 所示。



图 1.4 获取验证码

当用户单击“获取验证码”按钮后，需要等待 60 秒才能再次单击“重发验证码”按钮，

在这种情况下,就可以尝试用 `pointer-events` 实现禁用鼠标单击事件。在 `pointer-events` 属性中有一个 `none` 值,将 `pointer-events` 的值设置成 `none` 就不会响应鼠标事件了。举一个获取验证码的例子,代码如下。

```
<style>
  a{
    color:red;
  }
  .disable{
    pointer-events:none;
    color:#666;
  }
</style>
<a href="javascript:;" id="btn">发送验证码</a>
<script>
  var oBtn = document.getElementById('btn');
  oBtn.onclick = function(){
    oBtn.classList.add('disable');
    setTimeout(function(){
      oBtn.classList.remove('disable');
    },3000)
  };
</script>
```

如果看不懂这段代码也没关系,将这段代码复制下来即可。这段代码的意义就是定义了一个鼠标事件禁用的 `class`,单击“发送验证码”按钮后加上刚刚定义的 `.disable`,3秒以后再将这个 `class` 去掉。默认情况下的按钮,如图 1.5 所示。

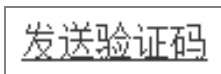


图 1.5 默认情况下的按钮

单击此按钮后,在3秒内不会再次响应单击事件。

`pointer-events` 除了可以实现此功能之外,还有很多用处,比如实现 `a` 标签禁止页面跳转,提高网页性能,用户在滚动页面时可能会不小心碰到一些元素上绑定的事件,这些事件就会被触发,从而浪费资源,但如果在页面滚动时给 `body` 加上 `pointer-events:none;` 属性,那么就避免了这个问题。

`pointer-events` 还有一个妙用, 比如将一个遮罩层元素的属性设置为 `pointer-events:none`, 这样就可以单击到遮罩层后面的内容, 如图 1.6 所示。

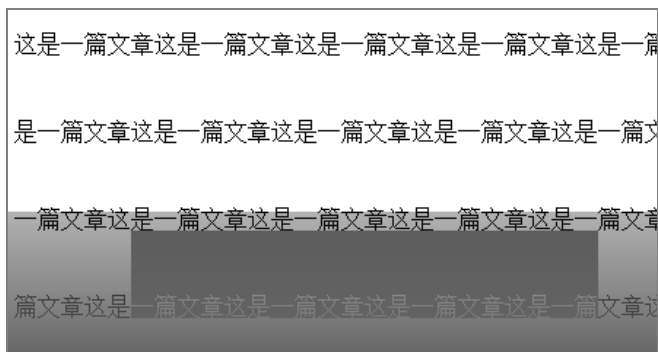


图 1.6 运用 `pointer-events` 后

如图 1.6 所示可以看到选中了遮罩层后面的内容, 但需要注意的是, `pointer-events:none` 只是用来禁用鼠标的事件, 通过其他方式绑定的事件还是会触发的, 比如键盘事件等。另外, 如果将一个元素的子元素 `pointer-events` 设置成其他值, 如 `auto`, 那么当单击子元素时, 还是会通过事件冒泡的形式触发父元素的事件。

1.2.2 玩转 CSS 选择器

(1) 当父元素只有一个子元素时会被选中, 代码如下。

```
<style>
  div:first-of-type:last-of-type{
    color:red;
  }
</style>
<div>123</div>
```

当只有一个 `div` 元素时, 效果如图 1.7 所示。当有多个 `div` 时不会被选中, 效果如图 1.8 所示。

123

图 1.7 当只有一个 `div` 时

123
456

图 1.8 当有多个 `div` 时

当然，更简单的方法是直接使用 CSS 3 中的结构性伪类选择器，当父元素只有一个子元素时会被选中，代码如下。

```
:only-child
```

(2) 当父元素有多个子元素时，选中第一个，代码如下。

```
<style>
  div:not(:last-of-type):first-of-type{
    color:red;
  }
</style>
<div>11111</div>
```

只有一个子元素时，不会被选中，效果如图 1.9 所示。当有多个子元素时，它会选中第一个，效果如图 1.10 所示。



图 1.9 只有一个子元素时



图 1.10 当有多个子元素时

当然，如果有多个子元素时，也可以选择其中任意一个子元素，但最后一个是被选中不了的，因为我们已经用“:not”否定了最后一个元素。如果想要摆脱这种限制，可以使用下面这种方案，代码如下。

```
:not(:only-child)
```

以有多个子元素时选中最后一个子元素为例，代码如下。

```
<style>
  div:not(:only-child):last-of-type{
    color:red;
  }
</style>
<div>11111</div>
<div>22222</div>
<div>33333</div>
```

当一个父元素有多个子元素时，最后一个元素会被选中，效果如图 1.11 所示。

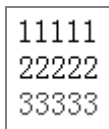


图 1.11 当有多个子元素时，选中最后一个元素

案例

合理使用这些选择器可以做很多事情，比如当只有一个子元素时，可以让它居中显示，如果有多个子元素时，可以让它水平排列，代码如下。

```
<style>
  .box div{
    width:100px;
    height:100px;
    border:1px solid red;
    margin:0 auto;
  }
  .box div:not(:only-child){
    float:left;
    margin-left:20px;
  }
</style>
<div class="box">
  <div></div>
</div>
```

当只有一个子元素时，这个 `div` 就会被居中显示，效果如图 1.12 所示。

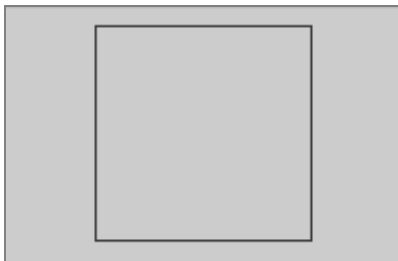


图 1.12 当只有一个子元素时居中显示

当有多个子元素时，效果如图 1.13 所示。

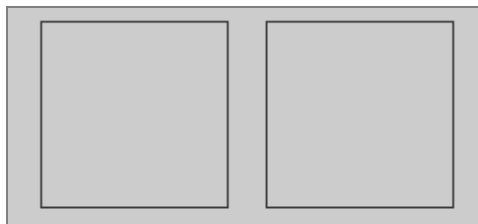


图 1.13 当有多个子元素时效果图

1.2.3 利用 padding 实现元素等比例缩放

padding 和 margin 有一个很奇怪的特点，它们的上下外边距的百分比是根据父元素的宽度来计算的。举个例子，代码如下。

```
<style>
  .box{
    width:100px;
    height:10px;
  }
  .box div{
    width:100%;
    padding-bottom:100%;
    background-color:red;
  }
</style>
<div class="box">
  <div></div>
</div>
```

效果如图 1.14 所示。

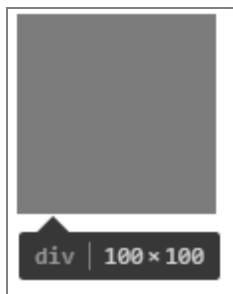


图 1.14 padding、margin 上下外边距的百分比

在此例子中可以看到 div 的宽度和高度都是 100px。如果根据父元素的高度来计算，那么 div 的高度最终应该是 10px，而不是 100px，因此，若需要一个等比例的元素，就可以利用这个特性，但如果使用这种方式，还需要解决另外一个问题，就是如果直接在子元素 div 中写入内容，那么高度会被“撑开”，那就不是等比例了。代码如下。

```
<div class="box">
  <div>padding-bottom</div>
</div>
```

若在 div 中加入一段文字，那么高度就不再是等比例了，效果如图 1.15 所示。



图 1.15 在 div 中加入一段文字后的高度

但是可以将代码进行修改，修改后的代码如下。

```
<style>
  .box{
    width:30%;
    height:10px;
  }
  .box div{
    position:relative;
    overflow:hidden;
    background-color:red;
  }
  .box div::after{
    content:'';
    display:block;
    padding-top:100%;
  }
```

```

.box div span{
    position:absolute;
    left:0;
    top:0;
}
</style>
<div class="box">
    <div>
        <span>图片</span>
    </div>
</div>

```

利用伪元素的 `padding-top` 来“撑开”父元素的高，内容通过绝对定位来使用，因为绝对定位的元素是不占位置的，这样一个等比例宽高缩放就完成了。有时这种特性很有用，比如针对下面这个需求，如图 1.16 所示。

现在需要将图片等比例缩放，也就是宽和高一样，但图片的宽度是自适应屏幕大小的，`img` 标签在只写宽度不写高度的情况下，高度会自适应宽度。图片没有加载出来之前的情况，如图 1.17 所示。



图 1.16 需求



图 1.17 图片没有加载出来

从图 1.17 可以看到在图片没有加载出来之前高度就没有了，这时利用 CSS 属性 `padding-top` 就可以解决这个问题，代码如下。

```

.photo a{
    position:relative;
    float:left;
    width: calc(33.33% - 1.6rem);
    margin:1.2rem 0 0 1.2rem;
}

```

```
        outline:1px solid #dedede;
    }
    .photo a::before{
        content:'';
        display:block;
        padding-top:100%;
    }
    .photo a img{
        position:absolute;
        left:0;
        top:0;
        width:100%;
        height:100%;
    }
```

使用一个伪元素将高度“撑起来”，而图片通过定位来做。还有一种更简单的做法，就是直接给 a 标签设置高度，单位使用 vw。vw 单位是相对于视口（屏幕）宽度的，代码如下。

```
.photo a{
    float:left;
    width: calc(33.33% - 1.6rem);
    height: calc(33.33vw - 1.6rem);
    margin:1.2rem 0 0 1.2rem;
    outline:1px solid #dedede;
}
.photo a img{
    display:block;
    width:100%;
    height:100%;
}
```

宽度怎么设置，高度就怎么设置，就是把百分比换成 vw。但是只在自适应方面才能这样用，如果是固定的宽、高，直接设置成一样的就行了，虽然 vw 可以实现，但兼容性还不是很不好。

1.2.4 calc 函数

在 CSS 中，如果需要用计算的功能，那么 calc 函数将非常有用。calc 函数允许进行任

何长度值的计算，运算符可以是加 (+)、减 (-)、乘 (*)、除 (/) 等。但需要注意的是，运算符前后都需要保留一个空格，虽然在某些特殊情况下可能不需要，但最好都加上，下面来介绍一些 calc 函数的使用场景。

场景一：

如图 1.18 所示，图中的内容一旦超过了浮动元素的高，那么这些文本就会与图片左对齐，这种效果并不是我们想要的。我们想要的效果，如图 1.19 所示。

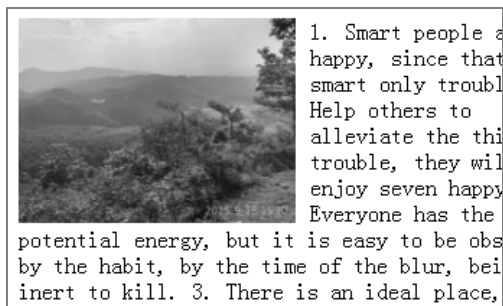


图 1.18 实际不理想的效果

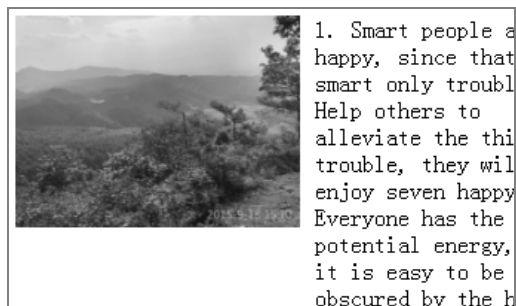


图 1.19 预期理想的效果

如果知道图片的宽度，那么解决这个问题也很简单，给这段文本添加一个左边距即可，但如果图片使用的是百分比，那么就无能为力了，而如果使用 calc 函数可以很好地解决这个问题，代码如下。

```
<style>
  .box img{
    width:50%;
    float:left;
  }
  .box p{
    margin-left:calc(50% + 10px);
  }
</style>
<div class="box">
  
  <p>.....</p>
</div>
```

利用 calc 函数更改代码后的效果如图 1.20 所示。

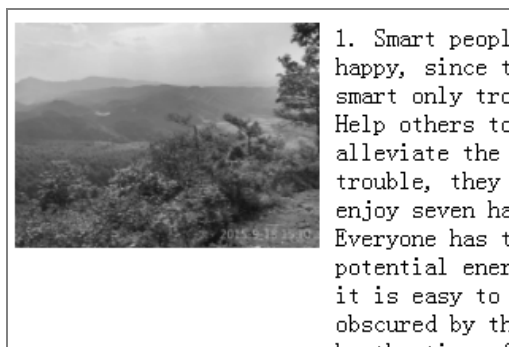


图 1.20 利用 calc 函数的效果

场景二：

有时使用百分比会出现一个问题，如图 1.21 所示。



图 1.21 使用百分比时可能会出现的问题

其中 CSS 代码如下。

```
<style>
  .box img{
    width:25%;
    margin:20px;
    float:left;
  }
</style>
```

导致这个问题出现的原因是使用了 `margin` 值，而代码中的 `width:25%` 并没有减去这个 `margin` 值。因此只需要用 `calc` 函数减去 `margin` 值就可以了，代码如下。

```
<style>
  .box img{
    width:calc(25% - 40px);
    margin:20px;
    float:left;
  }
</style>
```

最终效果如图 1.22 所示。



图 1.22 使用 `calc` 函数的最终效果图

场景三：

如果再结合媒体查询，那么就很容易实现一个响应式的布局，代码如下。

```
<style>
  .box img{
    width:calc(100% / 4 - 40px);
    margin:20px;
    float:left;
  }
  @media (max-width:600px){
    .box img{
      width:calc(100% / 2 - 40px);
    }
  }
</style>
```

这段代码表示在屏幕不小于 `600px` 时，一行最多可以放 4 张图片，如果屏幕小于或等于 `600px` 时，一行最多只能放两张图片。

1.3 隐藏元素

千万不要小看“隐藏”这个技能，多了解一点，就多一种选择。如果你是一个新手，就会发现在本节将出现很多你不认识的属性，它们可能是在 CSS 2 中就有的属性，也可能是在 CSS 3 中出现的新属性。

1. 通过设置 `width:0;`或 `height:0;`隐藏一个元素，代码如下。

```
div{width:0;}  
或  
div{height:0;}
```

一个物体是由宽和高组成的，那么至少这个物体得有宽和高，这种方式的缺点是隐藏不了文字。可以将元素的 `color` 设置成与背景色一样的颜色，这样就看不见了。也可以设置成透明色 (`transparent`)，但问题是它们的内容还是存在的，所以需要将文字的大小设置成 0，代码如下。

```
div{font-size:0;}
```

2. 将元素的 `opacity` 设置成 0，代码如下。

```
div{opacity:0;}
```

元素本身还在，只是看不见而已。

3. 通过定位将元素移出屏幕范围，代码如下。

```
div{  
    position:absolute;  
    left:-9999px;  
}
```

元素还在，只是超出了屏幕范围看不见了而已。

4. 通过 `text-indent` 实现隐藏文字的效果，代码如下。

```
div{text-indent:-99999px;}
```

给页面添加 LOGO 图片，若还想让搜索引擎搜索到，则需要添加这段文字，但如果又不想显示这段文字，就可以使用这个方法。

5. 通过 `z-index` 隐藏一个元素，代码如下。

```
<style>
```

```
.box{
    position:relative;
}
.box .item{
    position:absolute;
    left:0;
    top:0;
    width:100px;
    height:100px;
    border:1px solid red;
    z-index:-1;
}
.box .item:first-of-type{
    z-index:1;
}
</style>
<div class="box">
    <div class="item">程序员</div>
    <div class="item">设计师</div>
</div>
```

但你会发现文字被“透”上来了，效果如图 1.23 所示。



图 1.23 文字被“透”上来了

因为默认的背景设置是透明的，并且允许下面的元素“透”上来，想解决这个问题很简单，就是给元素添加一个背景，代码如下。

```
.box .item{
    position:absolute;
    left:0;
    top:0;
    width:100px;
    height:100px;
```

```
border:1px solid red;
background-color:#fff;
z-index:-1;
}
```

设置完成后，效果如图 1.24 所示。



图 1.24 添加背景后的效果

6. 通过给元素设置 `overflow` 来隐藏元素，代码如下。

```
div{
    width:100px;
    height:100px;
    overflow:hidden;
}
```

如果元素超出所设置的宽和高，溢出的部分就会被隐藏。如果想让整个元素隐藏，将元素的宽和高设置成 0 即可。经常通过这种方式将超出的文字隐藏，代码如下。

```
<style>
    h2{
        width:16ch;
        overflow:hidden;
        white-space:pre;
        text-overflow:ellipsis;
    }
</style>
<h2>享受一片宁静的天空。</h2>
```

当中文超出 7 个字符以后，文字就会被隐藏，效果如图 1.25 所示。

享受一片宁静的...

图 1.25 超出的文字被隐藏

7. 通过 `visibility` 将元素设置为不可见，代码如下。

```
div{visibility:hidden;}
```

虽然元素不可见，但还占位置。

8. 通过 `display` 将元素彻底隐藏，代码如下。

```
div{display:none;}
```

元素会被隐藏，并且不占着位置。

9. 将元素的背景设置为透明，字体大小设置为 0，代码如下。

```
div{
    font-size:0;
    background-color:transparent;
}
```

元素还在，只是看不见。

10. 将元素的 `max-width` 或 `max-height` 设置为 0，代码如下。

```
div{max-height:0;}
```

或

```
div{max-width:0;}
```

这样元素的宽度就只能是 0 了，但是还有文字溢出的问题，如图 1.26 所示。

尽管元素宽度是 0，但文字还是被显示出来了，若想解决这个问题，将文字大小设置成 0 就可以了，或者使用代码 `overflow:hidden`。如果你仔细看这个效果，会发现它实际上是一个文字竖排的效果，不过对于英文来说，还得设置一个换行属性，换行属性代码如下。

```
word-break:break-all;
<style>
    h2{
        max-width:0px;
        word-break:break-all;
    }
</style>
<h2>享受一片宁静的天空 AAA</h2>
```

效果如图 1.27 所示。

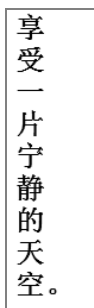


图 1.26 文字溢出

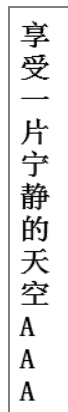


图 1.27 设置 word-break: break-all; 解决英文不换行问题

11. 通过 transform 的 translate 函数隐藏一个元素，代码如下。

```
div{transform:translate(-99999px);}
```

与 “left:-99999px;” 原理一样，将元素移出屏幕可视区。

12. 将元素的缩放设置成 0，代码如下。

```
transform:scale(0);
```

13. 让元素重叠，代码如下。

```
div{transform:skew(90deg);}
```

元素重叠了，类似 width 等于 0。

14. 设置 margin 为负值，代码如下。

```
div{margin-left:-999999px;}
```

将元素移出屏幕可视区。

15. 将元素裁剪，代码如下。

```
-webkit-clip-path:polygon(0px 0px,0px 0px,0px 0px,0px 0px);
```




第 2 章 CSS 核心概念

2.1 CSS 解析规则

每个语言都有其自己的一套解析规则，CSS 也不例外，如果你知道它的解析规则就可以减少出现 BUG 的次数。

因为 CSS 对空格不敏感，因此在每个样式后都要加一个分号，不然会把写在后面的样式当成一个整体来解析，直到遇见分号为止，代码如下。

```
<style>
  .box{
    width:100px
    height:100px;
    background-color:red;
  }
</style>
<div class="box">box</div>
```

如果没有加分号，结果如图 2.1 所示。



图 2.1 CSS 对空格不敏感

上面的代码会被浏览器解析成下面所示内容。

```
.box{
  width:100px height:100px;
  background-color:red;
}
```

不过最后一个属性的分号不是必须加的，代码如下。

```
.box{
  width:100px;
  height:100px;
  background-color:red
}
```

当遇见不认识的属性或值时，将忽略这个属性，继续解析后面的属性，代码如下。

```
<style>
  .box{
    5:100px;
    height:100r;
    border:1px solid red;
  }
</style>
<div class="box">box</div>
```

结果如图 2.2 所示。

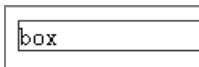


图 2.2 遇到不认识的属性或值时会被忽略

刚刚的代码会被浏览器解析为如下内容。

```
.box{
  border:1px solid red;
}
```

这也就是为什么在做浏览器兼容的时候可以像下面这样写：

```
-webkit-box-sizing:border-box;
box-sizing:border-box;
```

当浏览器不认识 `box-sizing` 属性而认识 `-webkit-box-sizing` 属性时就会使用 `-webkit-box-sizing`，而如果这两个属性都认识，那么 `box-sizing` 属性就会把 `-webkit-box-sizing` 属性给层叠掉。

3. 对于复合属性，只要其中一个值是错误的，那么整个属性都不解析，代码如下。

```
<style>
  .box{
    padding:10px i;
```

```

    border:1px solid red;
  }
</style>
<div class="box">box</div>

```

结果如图 2.3 所示。

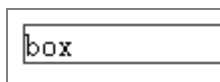


图 2.3 复合属性被忽略

刚刚的代码被浏览器解析为如下内容。

```

.box{
  border:1px solid red;
}

```

4. 最后一个 CSS 规则的大括号可以不闭合，代码如下。

```

<style>
  .box{
    color:orange;
  }
  .box{
    border:1px solid red;
</style>
<div class="box">box</div>

```

结果如图 2.4 所示。



图 2.4 大括号可以不闭合

刚刚的代码被浏览器解析为如下内容。

```

<style>
  .box{
    color:orange;
  }
  .box{
    border:1px solid red;

```

```

    }
</style>

```

5. 后代选择器中间必须加空格，代码如下。

```

<style>
    .box p{
        color:orange;
    }
</style>

```

6. 换行会被当作一个空格，代码如下。

```

<style>
    div
    .box{
        color:orange;
    }
    .box{
        border:1px solid red;
    }
<div class="box"><div class="box">box</div></div>
</style>

```

结果如图 2.5 所示。



图 2.5 换行被当作一个空格

刚刚的代码会被浏览器解析为如下内容。

```

<style>
    div .box{
        color:orange;
    }
    .box{
        border:1px solid red;
    }
</style>

```

7. 关键字不可以用引号，否则会被忽略，代码如下。

```
<style>
  .box{
    color:"orange";
  }
</style>
```

8. 在多组选择器中，只要有一个选择器是错误的，其他的也不会执行，代码如下。

```
<style>
  .box,0div{
    color:red;
  }
</style>
```

9. 在 CSS 中，如果需要使用到@import 规则导入外部样式表，那么需要注意的是，@import 规则只能写在其他样式规则的前面，否则@import 规则会被忽略，代码如下。

```
<style>
  .box{
    color:pink;
  }
  @import '1.css';
</style>
```

刚刚的代码是不会被执行的，改成如下代码即可。

```
<style>
  @import '1.css';
  .box{
    color:pink;
  }
</style>
```

之所以要把@import 规则写在其他样式规则的前面，主要是因为 CSS 的设计者害怕通过@import 规则导入的外部样式表会把其他的 CSS 样式规则层叠掉，毕竟现在大多数开发者喜欢模块化开发，把一份样式文件拆分成不同的子模块，但是这些子模块里面的样式难免会互相影响，因此谁层叠谁就显得格外重要，这也是为什么@import 需要写在其他样式规则前面的原因之一。另外，如果需要修改通过@import 规则导入过来的样式，直接在其后面做相应的修改就可以了。否则，只能修改相应的源文件。

2.2 替换元素与非替换元素

元素可分为替换元素和非替换元素。

- 替换元素：是指浏览器根据元素的标签和属性来决定元素的具体内容。例如“:”，浏览器会根据标签的 src 属性的值来读取图片信息并显示。例如“<input type="radio">:”，浏览器会根据 input 标签的 type 属性来决定是显示输入框还是单选按钮等。
- 非替换元素：其内容直接显示在客户端的元素叫非替换元素。例如如下内容：

```
<h1>是在叫我吗</h1>  
<div>干啥</div>
```

这些内容在源代码中会被直接显示出来。

2.3 属性值的计算规则

当客户端在解析文档并建构文档树之后，会给文档中每一个元素的属性分配一个属性值。这个属性的最终值可能会经过指定值、计算值、使用值、实际值这 4 个步骤。

(1) 指定值：客户端根据以下机制为每个属性分配指定值。

- 如果用户直接指定一个值，且该值不是 inherit，则使用这个值。否则进行下面一步。
- 如果用户指定的值是 inherit，并且这个元素不是根元素，则使用父元素的计算值。否则进行下面一步。
- 使用属性的初始值。

(2) 计算值：在某些情况下，指定的值需要进行计算处理，比如通过 em、rem 设置的值，或者背景图片地址上的 URL 地址需要被解析成绝对地址，子元素继承的就是这个值。在这个过程中客户端并不会渲染文档。

(3) 使用值：计算值会被尽可能地处理，而不渲染文档。但是有一些值只能在文档被渲染时才可以确定。比如一个元素的宽度被设置成包含块的百分比时，这个元素的宽度是不能被确定的，直到包含块的宽度被确定。所以有些值还需要使用以后才能计算出来。

(4) 实际值：在某些情况下，最终看到的效果会和设置的值有所不同，导致出现这种

情况的原因可能是因为客户端对设置的值无法进行精确计算，比如小数的位数过长，那么它可能会计算成一个近似的值。也可能是用户的设备不支持等，都会影响最终的值。

1. 继承

某些属性的子元素会继承父元素的值，代码如下。

```
<style>
  .box{
    color:red;
  }
</style>
<div class="box">
  <div class="item">item</div>
</div>
```

效果如图 2.6 所示。

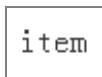


图 2.6 子元素继承父元素的值

因为颜色是可继承属性，且.item 没有设置颜色，所以继承了父元素.box 的颜色。

当存在继承时，子元素继承的是父元素的计算值，代码如下。

```
<style>
  .box{
    font-size:10px;
  }
  .item{
    font-size:120%;
  }
</style>
<div class="box">
  <div class="item">item</div>
</div>
```

如以上代码所示，.item 继承的是.box 计算出来的值，也就是 10px，用 10px 乘以 120%。

2. inherit 继承

可继承或不可继承的属性都可以通过 inherit 来继承父元素的某个属性，继承的值是父元素的计算值。如果对根元素设置 inherit，那么客户端会将该属性设置成初始值。以下是

一个 `inherit` 继承的例子，代码如下。

```
<style>
  .box{
    width:100px;
    height:100px;
    border:1px solid red;
  }
  .item{
    width:50px;
    height:inherit;
    border:inherit;
  }
</style>
<div class="box">
  <div class="item">item</div>
</div>
```

通过给 `.item` 设置 `inherit` 继承了父元素的高和边框，结果如图 2.7 所示。

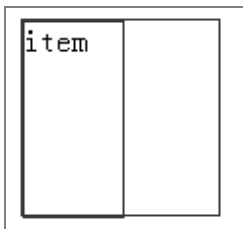


图 2.7 `inherit` 继承不可继承的值

2.4 可视化格式模型

学过 CSS 知识的大部分人都对盒子模型有所了解，盒子模型是由元素自身产生的，主要是对 `width`、`height`、`padding`、`border`、`margin` 的解释。而可视化格式模型则是把这些盒子按照一定的规则摆放到页面上，即规定每个盒子应该怎样去布局。可视化格式模型与一个元素的布局息息相关。

可视化格式模型规定了客户端（一般指浏览器）在媒介（屏幕、印刷、语音合成器、盲人设备等，一般指屏幕）中应该如何去处理文档树（在 HTML 中，每个元素由父元素、

子元素、祖先元素、兄弟元素等组成，这些元素组成的结构非常像树状，所以称它为文档树)。

在可视化格式模型中，假设每一个元素都会产生若干个矩形框，0 个或多个，那么这些矩形框的布局会受以下因素的影响。

- 框的大小以及框的类型：框的大小由作者指定或由客户端指定，框的类型是指由 `display` 所指定的值；
- 定位类型：定位类型分为 3 种，即常规流、浮动和定位；
- 矩形框之间相互影响：在文档中，如果一个元素前后有其他元素，那么这个元素可能会被其他元素的布局所影响。比如它前面是一个块元素，那么它会换行显示或使用 `margin`、浮动等，这些都会对这个元素产生一定的影响。另外也会受它所在的包含块影响；
- 外部因素：一个元素的布局还会受可视区域大小的影响，比如用浮动写的布局，一旦屏幕缩小它就会掉下来。又比如自适应布局，它会根据屏幕大小来自适应屏幕。另外，图片的尺寸也会影响元素的布局。

虽然可视化格式模型规定着一个盒子的布局，但它并不规定所有的布局细节，还有一部分由客户端自己决定（比如字间距等）。因此部分内容根据用户端的不同，结果可能会有所不同（比如一个盒子的布局，可能在 **Chrome** 浏览器中与在 **IE** 浏览器中看到的不一樣）。

视口 (viewport)

视口也叫可视窗口，一个盒子的布局往往也受可视窗口大小的影响，比如 **HTML** 默认的宽度就为可视区域的宽。如果此盒子的布局是基于可视窗口大小布局的（根据屏幕百分比布局等），那么当可视窗口改变时，布局也会改变。

当可视窗口的尺寸小于文档渲染的页面大小时，客户端应该提供滚动机制（滚动条）。每个页面只能有一个可视窗口，但客户端可以在一个可视窗口中同时渲染多个页面（比如用 `iframe` 嵌入的页面）。

2.5 包含块

在 **CSS** 中，有时一个元素的位置和尺寸的计算都相对于一个矩形，这个矩形被称作包含块。包含块是一个相对的概念，比如子元素的初始化布局总是在父元素的左上角，这就

是一个相对的概念。其中父元素就是一个参照物，但实际上这些元素都是根据包含块所在的位置进行布局。每个元素都会生成一个包含块，但这个包含块是虚无的，你看不到也摸不着，只是一个概念。举个例子，代码如下。

```
<div>
  <p>
    <span></span>
  </p>
</div>
```

div 包含着 p，p 包含着 span，此时 div 会生成一个包含块来包含 p 和 span，p 会生成一个包含块来包含 span。包含块最主要的作用就是给它里面的元素一个位置上的参照，也就是它应该从哪里开始摆放。

包含块并不会限制它里面元素的大小，如果里面的元素比包含块大，那么超出的部分就会被溢出，代码如下。

```
<style>
  div{
    width:100px;
    height:100px;
    background-color:red;
  }
  div p{
    width:100px;
    height:100px;
    margin-left:100px;
    background-color:pink;
  }
</style>
<div>
  <p></p>
</div>
```

效果如图 2.8 所示。



图 2.8 元素比包含块大

如上面这段代码，div 生成的包含块包含着 p，因此 p 元素参照着这个包含块来定位，可以看到代码中用了一个 `margin-left` 值，也是参照着包含块的位置来计算的。如果没有参照这个概念，不知道会乱成什么样子。另外，也可以发现虽然 p 元素超出了包含块，但并没有被隐藏，因此也可以说明包含块并不会对里面的盒子大小做限制。如果想让溢出的隐藏，只需要给 div 加一个 `overflow:hidden` 即可，代码如下。

```
div{  
  width:100px;  
  height:100px;  
  background-color:red;  
  overflow:hidden;  
}
```

效果如图 2.9 所示。

图 2.9 通过 `overflow:hidden` 隐藏溢出内容

一个元素的位置和尺寸与它的包含块息息相关，而元素会为它的子孙元素创建包含块，但这并不代表这个包含块就是它的父元素（不过这个父元素确实和包含块有着一些联系），再次强调包含块是一个概念。包含块最终的样子是由元素自身和它的祖先元素的样式等决定的（因此很容易把包含块理解成父元素）。为了更好地理解，在书中会用父元素代替包含块。

1. 包含块的创建

- 在 HTML 中，根元素的包含块叫作初始包含块，具体创建由客户端决定；
 - 当定位值为 `fixed`，则包含块由视口创建；
 - 当定位值为 `relative`、`static` 或没有设置定位属性，则包含块由最近的父元素或祖先元素创建；
 - 当定位值为 `absolute`，则包含块由最近的定位值 `relative`、`absolute`、`fixed` 创建。如果没有定位的祖先元素，则包含块为初始包含块（具体由用户端来决定）；
 - 当祖先元素是行内元素时，那么包含块取决于父元素或祖先元素的 `direction` 属性。
- 举个例子，代码如下。

```
<body>
  <div>
    <span>
      <em></em>
    </span>
  </div>
</body>
```

在这段代码中，`div` 的包含块是 `body` 创建的，`span` 的包含块又是 `div` 创建的，但这里并不是说 `div` 就是其包含块，`em` 的包含块由 `span` 创建。

2. 定位值为 `fixed`

当定位值为 `fixed` 时，结果如图 2.10 所示。



图 2.10 定位值为 `fixed`

可以看到元素贴在视口的最左边，代码如下。

```
<style>
  .box{
    position:fixed;
    left:0;
```

```

    top:0;
    width:100px;
    height:100px;
    background-color:pink;
  }
</style>
<div class="box">fixed</div>

```

当没有设置定位值或者定位值为 `relative`、`static` 时，结果如图 2.11 所示。

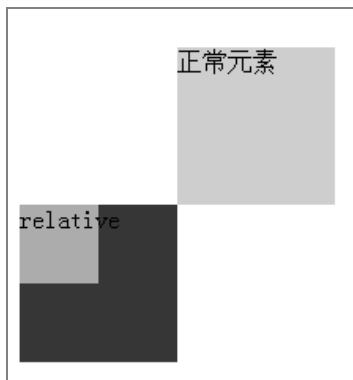


图 2.11 定位值为 `relative` 或 `static`

可以看到元素始终跟随包含块，代码如下。

```

<style>
  .box{
    width:100px;
    height:100px;
    margin:100px;
    background-color:pink;
  }
  .box2{
    position:relative;
    left:0;
    top:-100px;
    width:100px;
    height:100px;
    background-color:red;
  }

```

```
.box2 .item{
    width:50px;
    height:50px;
    background-color:orange;
}
</style>
<div class="box">
    <div class="item">正常元素</div>
</div>
<div class="box2">
    <div class="item">relative</div>
</div>
```

当定位值为 **absolute**，父元素没有设置定位时，包含块为初始包含块。而如果指定了其定位值，则包含块由其定位了的父元素或祖先元素创建，代码如下。

```
<style>
    .box{
        width:100px;
        height:100px;
        border:1px solid red;
    }
    .box1{
        position:absolute;
        left:0;
        top:0;
    }
    .box2{
        position:relative;
        left:100px;
    }
    .box2 .item,.box3,.box3 .item,.box4 .item{
        position:absolute;
    }
    .box4{
        position:fixed;
        left:200px;
    }
</style>
```

```

<div class="box box1">父元素或祖先元素没有定位时</div>
<div class="box box2">
  <div class="item">父元素或祖先元素定位值为 relative</div>
</div>
<div class="box box3">
  <div class="item">父元素或祖先元素定位值为 absolute</div>
</div>
<div class="box box4">
  <div class="item">父元素或祖先元素定位值为 fixed</div>
</div>

```

结果如图 2.12 所示。

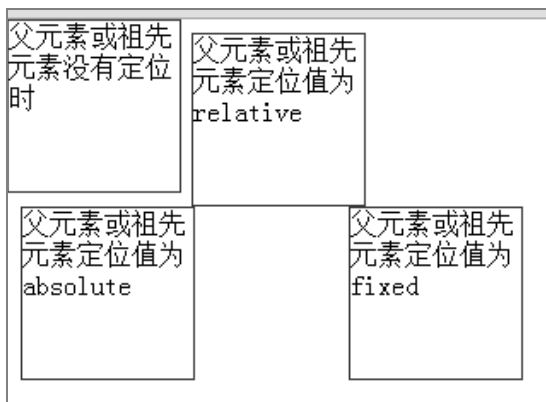


图 2.12 定位值为 absolute

当祖先元素是行内元素时，若 `direction` 值为 `ltr` 则右边补空白，若 `direction` 值为 `rtl` 则左边补空白，代码如下。

```

<style>
  .box{
    display:inline;
    direction:ltr;
    width:100px;
    height:100px;
  }
  .box2{
    display:inline;
    direction:rtl;
    width:100px;
  }
</style>

```

```

        height:100px;
    }
</style>
<div class="box">
    <div class="item">direction:ltr</div>
</div>
<div class="box2">
    <div class="item">direction:rtl</div>
</div>

```

结果如图 2.13 所示。

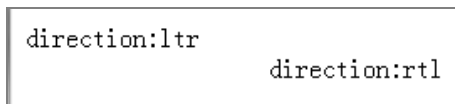


图 2.13 祖先元素是行内元素

2.6 控制框

本节将会讲解什么是块级元素、行内元素、块框、行内框、匿名行内框以及插入框。

1. 块级元素和块框

块级元素是那些在视觉上被格式化为块的元素（比如 `div`、`p`、`h1` 等）。另外通过 `display` 也可以将一个元素转变成块元素，其中可以产生块元素的值为 `block`、`list-item`、`run-in`、`table`。

块级元素除了 `table` 以外都会形成一个主块框，并且这个主块框只包含一种类型的框，就是说里面要么是块框，要么是行内框。主块框会为子孙元素建立包含块，生成内容。主块框参与块级格式化上下文（Block Formatting Context，简称 BFC）。某些块级元素还会在主块框之外生成额外的框，比如当某个元素的 `display` 值为 `list-item` 时，它会生成一个额外的框用来放置那些标志，比如 `li` 元素前面的标志。

2. 匿名块框

```

<div>
    这是一段<p>内容</p>
</div>

```

在上面的代码中，`div` 包括了一段文本和一个块框（`p`），此时 `div` 似乎既包括了行内框，又包括了一个块框，那么它会将所有的行内框都包含在一个匿名块框之中。

匿名块框的属性继承至最近的非匿名祖先盒（比如 color 等），不可继承的属性（比如 margin）则取其初始值。

3. 行内元素和行内框

行内元素是那些不生成新的内容块的元素，内容在行内显示，通过 display 也可以将一个元素转变成一个行内元素。其中可以产生行内元素的值为 inline、inline-table、run-in，行内元素会生成一个行内框。

4. 匿名行内框

```
<div>
  这是一段<em>内容</em>
</div>
```

在上面的代码中，div 包括了一段文本和一个 em 元素，此时 div 似乎包含的都是一个行内框，那么它会为这段文本生成一个匿名行内框。在格式化 table 时，会形成更多的匿名框。

匿名行内框的属性继承至最近的非匿名祖先盒（比如 color 等），不可继承的属性则取其初始值。空格内容会根据 white-space 特性被压缩，不会创建任何匿名行内框。不管是匿名行内框还是匿名块框，我们都可以称它为匿名框。

5. 插入框（run-in）

如果一个元素的 display 值为 run-in，那么它会根据后面的元素来确定它的类型应该是什么。

- 如果它后一个元素是行内元素或行内块元素，那么 display 值为 run-in 的元素将生成一个块框；
 - 如果它后一个元素是块级元素，那么 display 值为 run-in 的元素将生成一个行内框。
- 效果如图 2.14 所示。

图 2.14 插入框

代码如下。

```
<style>
  dt{
    display:run-in;
  }
```

```
</style>
<dl>
  <dt>CSS: </dt>
  <dd>Hello 你好。</dd>
</dl>
```

注意：Chrome 浏览器从 32 版本开始就将这个属性移除了，另外它也有兼容性的问题，但在 IE 8 和 IE 9 浏览器上测试都没有问题。

display 的一些特性如下所示。

- block 运用此值将生成一个块框；
- inline-block 运用此值将生成一个块框，元素内部按照块框格式化，但元素本身按照一个行内元素的形式来格式化（初始化）；
- inline 运用此值将生成一个或多个行内框；
- list-item 运用此值将生成一个块框和一个列表项行内框；
- none 运用此值将不在结构中显示，不产生任何框，并且子孙元素也不产生任何框；
- run-in 运用此值将根据后一个元素来选择要生成的框。

2.7 格式化上下文 BFC、IFC

元素所处的环境以及初始化就叫作格式化上下文。所谓格式化就是初始化（比如手机的重置、默认样式等），环境也称上下文（不同的环境会有不同的结果）。

格式化上下文的环境分为块格式化上下文和行内格式化上下文，而这些环境又影响着这些元素的初始化布局（也就是它们默认应该是什么样子）。块框参与块格式化上下文，行内框参与行内格式化上下文。

2.7.1 从 overflow 清除浮动看 BFC（块格式化上下文）

在项目中有不少地方会用到 overflow，其有如下几个属性。

- visible：不对超出的内容做处理；
- hidden：剪切超出的内容；
- scroll：不管内容是否超出，始终显示滚动条；

- auto: 超出的内容以卷动滚动条的方式呈现。

下面是一个关于 overflow 的应用，代码如下。

```
<style>
  .box{
    overflow:hidden;
  }
  .item{
    float:left;
    width:100px;
    height:100px;
    background-color:pink;
  }
</style>
<div class="box">
  <div class="item">Hello overflow</div>
</div>
```

效果如图 2.15 所示。



图 2.15 overflow 清除浮动

这是一段常用的代码，之所以给 .box 添加 overflow，是因为它的子元素 .item 浮动了，导致父元素无法自适应子元素的高度，所以给父元素加了一个 overflow。

1. overflow 的秘密

overflow 是对超出的内容做处理，如果一个元素没有设置宽和高，那么对它怎么处理呢？这就是问题所在，先不说 BFC，从另一方面说，overflow 是这样对它做处理的：如果元素没有设置宽和高，那么就将盒子里所有元素加起来的高度作为父盒子的高度。但这是一个很“搞笑”的事，因为把所有元素加起来作为父元素的高度，那么内容永远不可能超出。既然如此，是不是就说明当父元素没有设置宽和高时，不管父元素设置 overflow 的值为 hidden，还是 scroll 或 auto，它们的结果都是一样的呢？即永远不可能溢出。下面举两个例子。

设置 overflow 的值为 auto 的效果如图 2.16 所示。

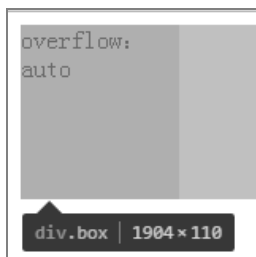


图 2.16 设置为 auto

代码如下。

```
<style>
  .box{
    overflow:auto;
  }
  .item{
    float:left;
    width:100px;
    height:100px;
    padding-bottom:10px;
    background-color:pink;
  }
</style>
<div class="box">
  <div class="item">overflow: auto</div>
</div>
```

设置 overflow 的值为 scroll 的效果如图 2.17 所示。



图 2.17 设置为 scroll

它出现滚动条很正常，但内容并没有溢出，代码如下。

```
<style>
  .box{
    width:120px;
    overflow:scroll;
  }
  .item{
    float:left;
    width:100px;
    height:100px;
    padding-bottom:10px;
    background-color:pink;
  }
</style>
<div class="box">
  <div class="item">overflow: scroll</div>
</div>
```

虽然 `scroll` 也可以达到清除浮动的效果，但因为默认会有一个滚动条，所以一般也不会用这个属性来清除浮动。不过既然知道了它的原理，也就不能说 `overflow` 是清除浮动的了，其不过是让父元素重新自适应了子元素的高度，顺便清除浮动。

2. 从 overflow 看 BFC

说到 `overflow`，不免要联系到 BFC。先来看一张图，如图 2.18 所示。



图 2.18 后面的元素被浮动元素盖住了

从图 2.18 可以看到浮动的某个元素把后面的元素盖住了，导致出现这个问题的原因是浮动的元素不占位置，所以后面的元素跑到浮动元素的位置上了，代码如下。

```
<style>
  div{
    width:100px;
    height:100px;
  }
  .box1{
    float:left;
    background-color:pink;
  }
  .box2{
    background-color:orange;
  }
</style>
<div class="box1">box1</div>
<div class="box2">box2</div>
```

如果给 .box2 加一个 `overflow:hidden;`，如图 2.19 所示。代码如下。

```
.box2{
  overflow:hidden;
  background-color:orange;
}
```

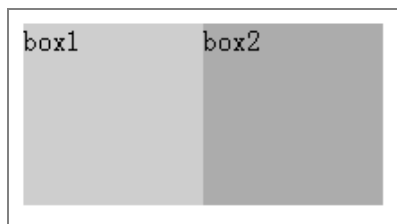


图 2.19 加上 `overflow:hidden;`

从现在的角度来看，应该会产生一个问题，那就是 `overflow` 真的只是对超出内容做处理这么简单吗？答案是否定的，因为它创建了一个 BFC。

2.7.2 块级格式化上下文 BFC

BFC 是块格式化上下文（Block Formatting Context）的简称，BFC 决定着块框的布局方式。

1. BFC 的特点

- 在一个块格式化上下文中，块框会朝着垂直方向一个接一个地排列，从包含块的顶部开始。如果两个及以上的框在同一个块格式化上下文中，那么它们相邻垂直方向的 `margin` 会合并成一个；
- 在一个块格式化上下文中，盒子从包含块的最左边摆放，即使存在浮动也还是靠在最左边。如果该元素建立了一个新的块格式化上下文（那么该盒子自身的宽度会因为浮动而变窄，变窄的宽度正好是浮动元素的宽度）；
- 在一个块格式化上下文中，包含块里面的所有元素，但不包含创建了新块格式化上下文元素；
- 当一个框创建了一个块格式化上下文时，它将包括浮动元素（这也就是为什么给父元素设置 `overflow:hidden` 可以自适应浮动元素的原因了）。

在出现下列情况时，将会创建一个块格式化上下文。

- 根元素（`html`）；
- 浮动 `float` 不为 `none` 的元素；
- 绝对定位元素 `position` 值为 `absolute`、`fixed` 的元素；
- `display` 值不为 `block` 的元素；
- 表格单元格 `display` 为 `table-cell` 的元素；
- 表格标题 `display` 为 `table-caption` 的元素；
- `overflow` 值不为 `visible` 的元素；
- 弹性盒子 `display` 为 `flex`、`inline-flex` 的元素。

2. 块级格式化上下文 BFC 详解

在一个块格式化上下文中，`display` 值为 `block` 的元素会朝着垂直方向一个接一个地排列，从包含块的顶部开始，如果两个及以上的元素在同一个块格式化上下文中，那么它们相邻垂直方向的 `margin` 会合并成一个，值由大的那个决定。如图 2.20 所示。



图 2.20 外边距合并

从图 2.20 可以看到 `.item1` 元素从包含块 `.box` 的顶部开始摆放，它和 `.item2` 垂直排列。另外可以看到 `.item2` 和 `.item1` 相邻垂直方向的 `margin` 合并了，代码如下。

```
<style>
  .box{
    width:100px;
    background-color:green;
  }
  .box > div{
    width:100px;
    height:100px;
  }
  .item1{
    margin-bottom:50px;
    background-color:pink;
  }
```



```

.item2{
    margin-top:100px;
    background-color:orange;
}
</style>
<div class="box">
    <div class="item1">item1</div>
    <div class="item2">item2</div>
</div>

```

在一个块格式化上下文中，盒子从包含块的最左边摆放。即使存在浮动，也还是靠在最左边。如果该元素建立了一个新的块格式化上下文，那么该盒子自身的宽度会因为浮动而变窄，变窄的宽度正好是浮动元素的宽度。

子元素从包含块的最左边摆放，即使存在浮动，也还是靠在最左边，如图 2.21 所示。

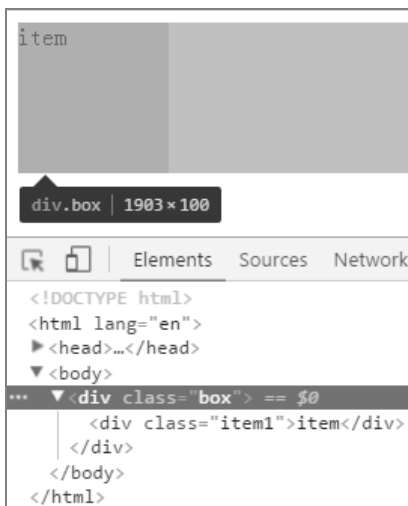


图 2.21 子元素从包含块的最左边摆放

可以看到即使上一个元素.item1 浮动了，但.item2 还是靠在了包含块的最左边，代码如下。

```

<style>
    .item1{
        float:left;
        width:100px;
        height:100px;
    }

```

```

        margin-left:10px;
        background-color:pink;
    }
    .item2{
        width:200px;
        height:100px;
        background-color:orange;
    }
</style>
<div class="box">
    <div class="item1">float</div>
    <div class="item2">block</div>
</div>

```

如果该元素建立了一个新的块格式化上下文，那么该盒子自身的宽度会因为浮动而变窄，变窄的宽度正好是浮动元素的宽度，如图 2.22 所示。

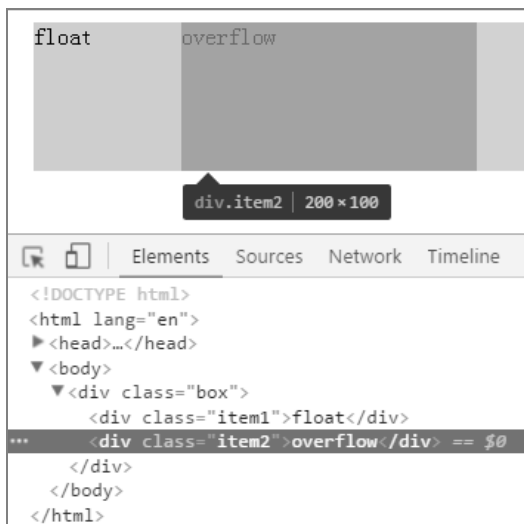


图 2.22 该盒子自身的宽度会因为浮动而变窄

这里给.item2 使用 `overflow:hidden` 创建了一个块格式化上下文，可以看到它的宽度变窄了，变窄的宽度正好是浮动元素的宽度，代码如下。

```

<style>
    .item1{
        float:left;

```

```

        width:100px;
        height:100px;
        background-color:pink;
    }
    .item2{
        overflow:hidden;
        width:200px;
        height:100px;
        background-color:orange;
    }
</style>
<div class="box">
    <div class="item1">float</div>
    <div class="item2">overflow</div>
</div>

```

这个特性可以用来实现如图 2.23 所示的效果。



图 2.23 右侧宽高自适应，左侧固定

图 2.23 所示的效果是右侧宽高自适应，并且左侧固定。但如果不做处理，在默认情况下的效果如图 2.24 所示。

之前的处理是给没有浮动的那个元素加一个 `margin-left` 或 `padding-left`，但学会 BFC 以后就可以不那么操作，只需要给没有浮动的那个元素创建一个 BFC 就可以，如图 2.25 所示。

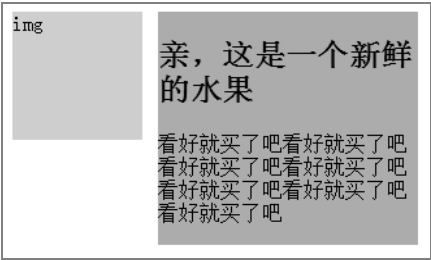


图 2.26 给浮动元素加 margin-right

代码如下。

```
.item2{
    overflow:hidden;
    width:200px;
    margin-left:112px;
    background-color:orange;
}
```

虽然 BFC 可以解决这个问题，但这种效果如果用 display:inline-block，可能就会出现问
题。不是它不能创建块格式化上下文，而是如果这个元素没有定义宽度，那么内容太长时
就会“掉下去”，如图 2.27 所示。



图 2.27 使用 display:inline-block

所以在这种自适应的情况下，还是不要用 display:inline-block 这种方式。

在一个块格式化上下文中，包含块包含所有元素，但不包含创建了新块格式化上下文的
元素，如图 2.28 所示。

当子元素使用 float 时，因为脱离标准流，所以包含块不再包含它，代码如下。

```
<style>
  .item{
    float:left;
    width:100px;
    height:100px;
    background-color:pink;
  }
</style>
<div class="box">
  <div class="item">把我收了吧</div>
</div>
```

当一个父元素创建了一个块格式化上下文时，它将包含浮动元素，如图 2.29 所示。



图 2.28 包含块不包含创建了新块格式化上下文的元素



图 2.29 包含浮动元素

这里用绝对定位给 .item 的父元素创建了一个块格式化上下文，所以包含块包含了它，代码如下。

```
<style>
  .box{
    position:absolute;
  }
  .item{
    float:left;
  }
```

```

        width:100px;
        height:100px;
        background-color:pink;
    }
</style>
<div class="box">
    <div class="item">把我收了吧</div>
</div>

```

另外, 根元素 `html` 也会创建一个块格式化上下文, 其也会自适应其浮动元素, 如图 2.30 所示。

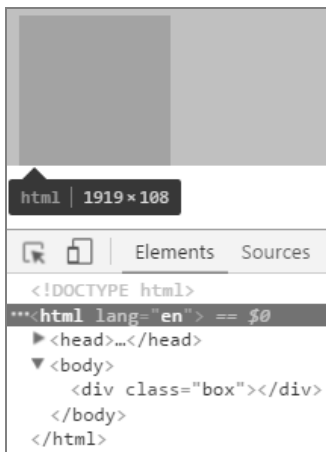


图 2.30 根元素创建一个块格式化上下文

代码如下。

```

<style>
    .box{
        float:left;
        width:100px;
        height:100px;
        background-color:orange;
    }
</style>
<div class="box"></div>

```

2.7.3 折叠外边距

在 CSS 中如果有两个及两个以上的盒子（可以是兄弟元素，也可以不是），那么它们之间的相邻外边距可以被合并成一个外边距。这种合并的外边距称为折叠外边距。

（1）根元素 html 的外边距不折叠，如图 2.31 所示。

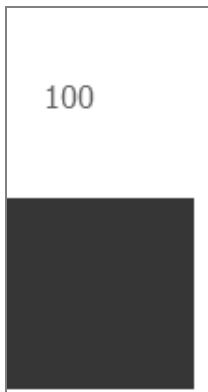


图 2.31 根元素 html 的外边距不折叠

代码如下。

```
<style>
  body,div{
    margin:0;
  }
  html{
    margin-top:50px;
  }
  div{
    width:100px;
    height:100px;
    margin-top:50px;
    background-color:red;
  }
</style>
<div></div>
```

（2）不用必须是兄弟元素，子元素和父元素垂直方向的外边距也会折叠，如图 2.32 所示。

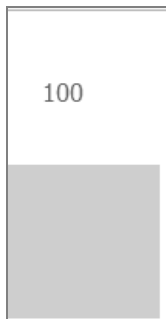


图 2.32 子元素和父元素垂直方向的外边距也会折叠

代码如下。

```
<style>
  body,div{
    margin:0;
  }
  div{
    width:100px;
    height:100px;
  }
  .box{
    margin-top:50px;
    background-color:red;
  }
  .item{
    margin-top:100px;
    background-color:pink;
  }
</style>
<div class="box">
  <div class="item"></div>
</div>
```

(3) 水平外边距不会合并。

(4) 当有多个子元素时，只有第一个元素的 `margin-top` 以及最后一个元素的 `margin-bottom` 会和父元素外边距折叠（注意：元素没有宽和高的情况除外），如图 2.33 所示。

代码如下。

```
<style>
  body,div{
```

```

        margin:0;
    }
    div{
        width:100px;
        height:100px;
    }
    .box{
        margin-top:50px;
    }
    .item1{
        margin-top:100px;
        margin-bottom:50px;
        background-color:pink;
    }
    .item2{
        margin-top:100px;
        background-color:green;
    }
</style>
<div class="box">
    <div class="item1"></div>
    <div class="item2"></div>
</div>

```

(5) 如果子元素同时是另外一个元素的父元素，那么同样会出现外边距折叠的情况，如图 2.34 所示。



图 2.33 相邻兄弟折叠

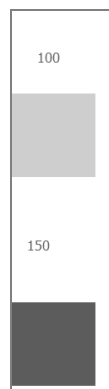


图 2.34 子元素同时是另外一个元素的父元素的外边距折叠

代码如下。

```
<style>
  body,div{
    margin:0;
  }
  div{
    width:100px;
    height:100px;
  }
  .box{
    margin-top:50px;
    background-color:red;
  }
  .item1{
    margin-top:100px;
    margin-bottom:50px;
    background-color:pink;
  }
  .item2{
    margin-top:100px;
    background-color:pink;
  }
  .child{
    margin-top:150px;
    background-color:green;
  }
</style>
<div class="box">
  <div class="item1"></div>
  <div class="item2">
    <div class="child"></div>
  </div>
</div>
```

(6) 负的外边距折叠，会从正的相邻外边距的最大值中扣除负的相邻外边距的绝对值的最大值，如图 2.35 所示。

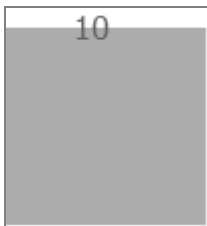


图 2.35 负的外边距折叠

代码如下。

```
<style>
  body,div{
    margin:0;
  }
  div{
    width:100px;
    height:100px;
  }
  .box{
    margin-top:60px;
    background-color:red;
  }
  .item{
    margin-top:-50px;
    background-color:orange;
  }
</style>
<div class="box">
  <div class="item"></div>
</div>
60-50 = 10
```

(7) 创建了新块格式化上下文的元素不会出现外边距折叠现象，可以创建块格式化上下文的属性，包括浮动、定位等。

2.7.4 行内格式化上下文 IFC

行内格式化上下文，简称 IFC，行内框参与行内格式化上下文，行内框中有一个行框的概念。

1. 什么是行框

在行内格式化上下文中，每个框都是一个接一个地水平排列，起点是包含块的顶部。水平方向上的 `margin`、`border` 和 `padding` 在框之间得到保留。框在垂直方向上可以以不同的方式对齐，通过 `vertical` 设置。框的每一行叫作行框，如图 2.36 所示。

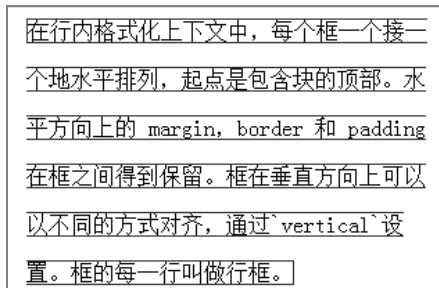


图 2.36 行框

行框的宽度由它的包含块和其中的浮动元素决定，高度由行高决定。

2. 行内框

- 如果几个行内框（宽度超出行框）在水平方向无法放入到一个行框中，那么它们将可能被分割成多个行框，来包含这些行内框，每一行就是一个行框。另外也有可能由于双向文本的处理而将一个行框分割成多个行框；
- 如果一个行框不能被分割（比如行内框只包含单个字符，或被设置成不允许换行等），那么行内框将会溢出；
- 在分割处不会运用 `margin`、`padding`、`border` 等属性。

下面举一个例子，代码如下。

```
<style>
  div{
    width:50px;
  }
  span{
    border:1px solid red;
  }
</style>
<div>
  <span>放不下就下一行 aaaaaaaaaaaaaaaaaaaaaa。</span>
</div>
```

效果如图 2.37 所示。

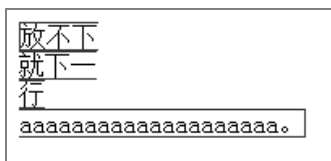


图 2.37 分割处不会被运用某些属性

如图 2.37 所示，分割处的行框并没有被运用边框样式。另外，因为图中的字母 a 是连在一起的，默认是不换行的，所以内容被溢出了。

当行内框的高度小于包含它的行框时，行内框可以通过 `vertical-align` 来设置行内框在行框中垂直方向的对齐方式，默认为基线（baseline）对齐。

举个例子，代码如下。

```
<style>
  span:nth-of-type(1){
    font-size:50px;
    border:1px solid orange;
  }
  span:nth-of-type(2){
    vertical-align:top;
  }
</style>
<span>我是来捣乱的</span>
<span>垂直对齐</span>
```

效果如图 2.38 所示。

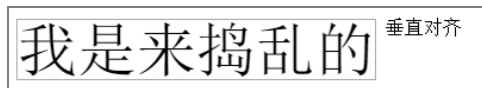


图 2.38 通过 `vertical-align` 来设置行内框在行框中垂直方向的对齐方式

如以上代码所示，两个 `span` 都在一个行框中，因为第一个 `span` 将行框的高度撑大了。因此对于第二个 `span` 就有一部分是空白的，那么就可以用 `vertical-align` 属性来设置行内框的垂直对齐方式。尽管行框和行内框高度一样时也可以设置，但没有任何意义。

当行内框宽度的总和小于包含它们的行框时，可以通过 `text-align` 来设置水平对齐，代码如下。

```
<style>
  p{
    width:100px;
    text-align:center;
    border:1px solid #ddd;
  }
</style>
<p>水平</p>
```

效果如图 2.39 所示。

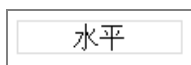


图 2.39 text-align 设置水平对齐

如果一个块元素中只有文本，那么里面的文本也是被匿名行内框包含的。

2.7.5 行高的计算

在行内格式化中，客户端会将行内框排列为垂直方向的行框的堆叠。行框的高度由以下特点决定。

- 计算行框内每个行内框的高度；
- 行内框按照 vertical-align 属性在垂直方向对齐；
- 行框的高度是最上面的框的顶边到最下面的框的底边的距离相加的和。

需要注意的是，空的行内框也包括边距、边白、边框和行高，因此与那些有内容的元素一样，也会影响计算，代码如下。

```
<style>
  .box{
    width:100px;
    border:1px solid red;
  }
  span{
    background-color:pink;
  }
  span:last-of-type{
    line-height:50px;
  }
</style>
```

```

</style>
<div class="box">
    <span>这是一个行内框吗</span>
    <span></span>
</div>

```

效果如图 2.40 所示。

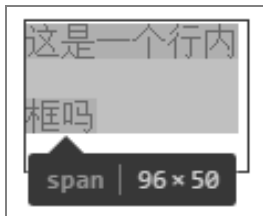


图 2.40 空的行内框也会影响行框的计算

由于一个行内框的高度和文字的高度可能不太一样，因此在文字的上方或下方可能有空白，如图 2.41 所示。

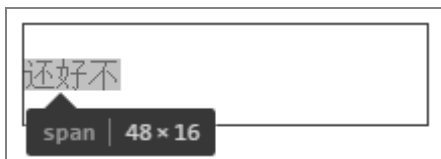


图 2.41 文字的上方或下方可能有空白

可以看到文字的高度小于行高的高度，因此有很多空白，客户端处理的方式是将文字放置在行内框的垂直中央。因此可以看到它垂直居中了，但如果父元素设置了高度，行高必须和父元素的高度一样才能垂直居中。以上效果的代码如下。

```

<style>
    .box{
        width:200px;
        border:1px solid red;
    }
    span{
        line-height:50px;
    }
</style>
<div class="box">

```



```
<span>还好不</span>
</div>
```

如果 `line-height` 的值小于字体尺寸，那么行内框的高度将小于字体尺寸，并且会将字体渗出到框的外面，如图 2.42 所示。

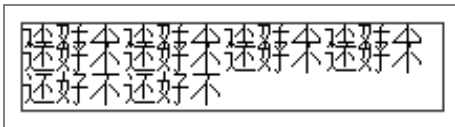


图 2.42 字体渗出到框的外面

以上是在 IE 7 浏览器中显示的结果，其他浏览器与此处理方式不一样，如果 `line-height` 小于字体尺寸，那么不做处理，以上效果的代码如下。

```
<style>
  .box{
    width:200px;
    border:1px solid red;
  }
  span{
    line-height:10px;
  }
</style>
<div class="box">
  <span>还好不还好不还好不还好不还好不还好不还好不还好不还好不还好不</span>
</div>
```

在 Chrome 浏览器上的效果如图 2.43 所示。虽然非替换元素的边距、边框和边白并不加入行内框高度的计算，但它们还是在行内框周围得到渲染。这就意味着，如果一个行框的高度小于框的外边，背景、边白和边框的颜色就有可能渗入到相邻的行框中，如图 2.44 所示。

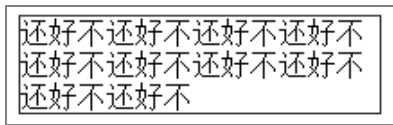


图 2.43 在 Chrome 浏览器中的效果

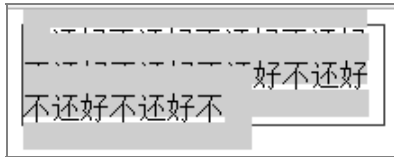


图 2.44 背景等可能被渗入到相邻的行框中

代码如下。

```
<style>
  .box{
    width:200px;
    border:1px solid red;
  }
  span{
    line-height:10px;
    border:15px solid pink;
  }
</style>
<div class="box">
  <span>还好不还好不还好不还好不还好不还好不还好不还好不还好不</span>
</div>
```

有些客户端可能使用行框来裁剪边框和边白区域。在 IE 7 浏览器下，当边框超出包含块时会被裁剪，如图 2.45 所示。

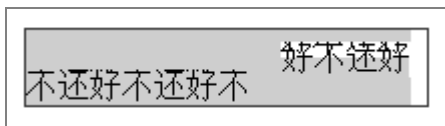


图 2.45 在 IE 7 浏览器下当边框超出包含块时会被裁剪

如果将 `line-height` 设置在一个块元素中，那么它相当于为里面的每一个行内框指定了最小高度，如图 2.46 所示。

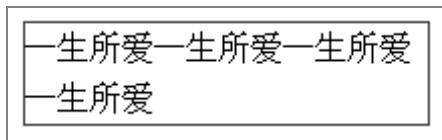


图 2.46 为每一个行内框指定了最小高度

代码如下。

```
<style>
  .box{
    width:200px;
    line-height:25px;
    border:1px solid red;
  }
```

```

    }
</style>
<div class="box">
    <span>一生所爱一生所爱一生所爱一生所爱</span>
</div>

```

如果将 `line-height` 设置在一个行内元素中，那么它指定了该元素生成的每个框的确切高度（除了行内替换元素以外，它的框高度由 `height` 属性决定），如图 2.47 所示。



图 2.47 每个框的确切高度

代码如下。

```

<style>
    .box{
        width:200px;
        border:1px solid red;
    }
    span{
        line-height:23px;
    }
</style>
<div class="box">
    <span>一生所爱</span>
</div>

```

`line-height` 具有以下一些属性。

- **normal:** 如果设置此值，则由客户端指定具体的值。在一般情况下，客户端会根据字体的尺寸设置一个合理的值；
- **length:** 将行高设置成具体的值，不允许是负值；
- **number:** 设置成字体的倍数；
- **percentage:** 字体的百分比。

以下所示的 4 个设置结果都是一样的。

```

span{font-size:15px;line-height:2;}
span{font-size:15px;line-height:200%;}

```

```
span{font-size:15px;line-height:2em;}
span{font-size:15px;line-height:30px;}
```

效果如图 2.48 所示。



图 2.48 相同效果

如果一个元素包含不止一个字体渲染的文本，那么客户端则根据最大的字体尺寸确定 line-height 的值，代码如下。

```
<style>
  .box{
    width:200px;
  }
  em{
    font-style:normal;
    font-size:25px;
  }
</style>
<div class="box">
  <span>一生所爱<em>这</em></span>
</div>
```

效果如图 2.49 所示。



图 2.49 根据最大的字体尺寸确定 line-height 的值

明明设置的是 25，最终运行结果却比 25 大，那为什么 line-height 的值是 29 呢？出现这种情况是因为在没有设置行高时，默认行高的值是 normal，而 normal 是由客户端指定的，所以不一定和字体大小一模一样。



第 3 章 CSS 单位究竟来自何方

3.1 百分比究竟为谁

在 CSS 中经常使用“100%”，但它究竟是根据什么来计算的呢？
举一个例子，代码如下。

```
<style>
  div{
    width:100%;
    height:100px;
    background-color:pink;
  }
</style>
<div>CSS 100%</div>
```

在使用百分比时必定有一个参照物，那么这里的百分比相对的是谁呢？代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  body{
    width:100px;
  }
  div{
    width:100%;
    height:100px;
    background-color:pink;
  }
</style>
```

```
</style>  
<div>CSS 100%</div>
```

结果如图 3.1 所示。

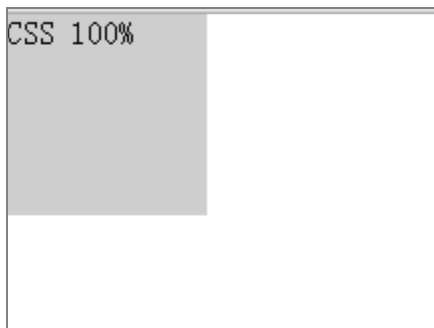


图 3.1 百分比参照物

这里给 **body** 加了一个宽度，发现 **div** 由原来的全屏宽度变成了与 **body** 一样的宽度，难道“100%”是根据 **body** 来计算的？代码如下。

```
<style>  
  body,div{  
    margin:0;  
    padding:0;  
  }  
  body{  
    width:100px;  
  }  
  .box{  
    width:200px;  
  }  
  .box .item{  
    width:100%;  
    height:100px;  
    background-color:pink;  
  }  
</style>  
<div class="box">  
  <div class="item">item 100%</div>  
</div>
```

结果如图 3.2 所示。

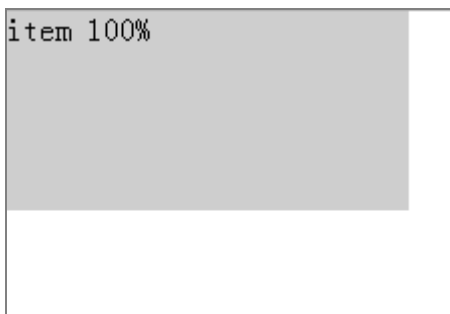


图 3.2 根据父元素计算结果

可以看到 `item` 并没有根据 `body` 计算，而是根据它的父元素计算的。在上一个例子中根据 `body` 计算是因为 `body` 就是它的父元素。

严格来说，子元素通常将继承父元素计算过的值当作百分比的参照，对于不可继承的属性和根元素，则使用初始值作为参照。

如果它的父元素没有设置宽度又会是什么情况呢？代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  body{
    width:100px;
  }
  .box .item{
    width:100%;
    height:100px;
    background-color:pink;
  }
</style>
<div class="box">
  <div class="item">item 100</div>
</div>
```

结果如图 3.3 所示。

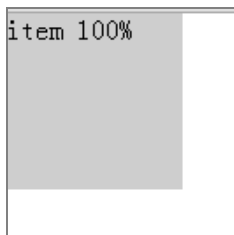


图 3.3 父元素没有设置宽度的情况

虽然.box 没有设置宽度，但默认继承了 body 计算过的值，又因为.box 是.item 的父元素。因此.item 又继承了.box 计算过的值，所以才出现了上面这种情况。当一个块级元素不设置宽度时，则它的宽度默认为全屏，就是因为它继承了包含块的宽度。

一般用百分比设置宽度，但在设置高度时需要注意，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:100px;
    height:100%;
    background-color:pink;
  }
</style>
<div class="box">height 100%</div>
```

结果如图 3.4 所示。

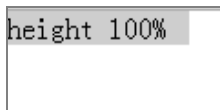


图 3.4 高度参数设置为 100%时没有效果

可以看到当.box 的高度设置为 100%时，没有效果，首先想到的是 body 的问题，于是将 body 的高度设置为 100%，代码如下。

```
body{
  height:100%;
}
```


结果如图 3.5 所示。

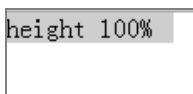


图 3.5 将 body 高度设置为 100%

可以看到将 body 的高度设置为 100% 也没有用，用 Chrome 浏览器可以看出“破绽”，如图 3.6 所示。

那么这时可以肯定是 html 的问题，如图 3.7 所示。

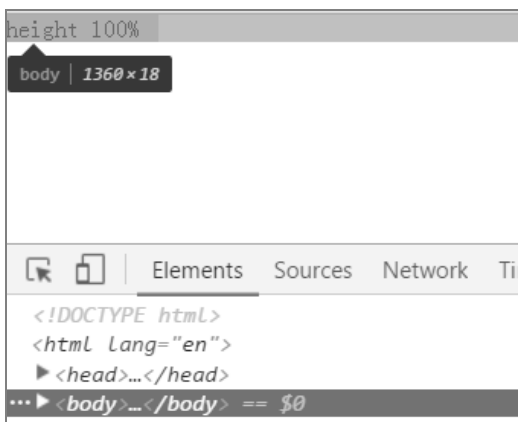


图 3.6 在 Chrome 浏览器中查看

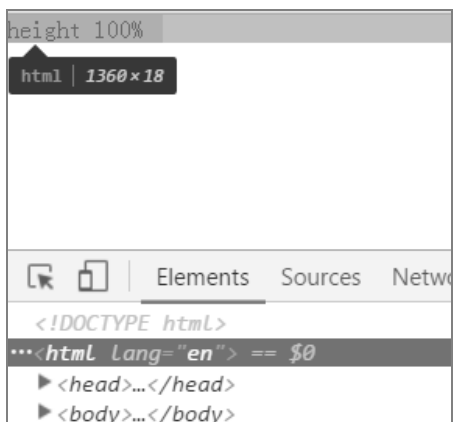


图 3.7 html 的问题

果然是 html 的问题，为什么上面显示“18px”呢？先来看下面这个例子，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:100px;
    height:100px;
    background-color:pink;
  }
</style>
<div class="box">height 100%</div>
```

结果如图 3.8 所示。

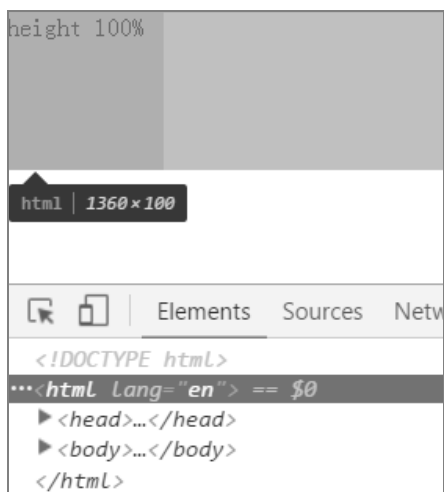


图 3.8 html 也是 100px

如图 3.8 可以看出，html 竟然也是 100px。那么可以推断出父元素在不设置高度的情况下，是自适应子元素高度的（在不设置高度的情况下，html 和 body 的高度是所有内容加起来的高度），如果父元素设置了高度，则是另外一种情况了。那么前面为什么是 18px 呢？代码如下。

```
<style>
  html,body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:200px;
    height:100%;
    line-height:25px;
    background-color:pink;
  }
</style>
<div class="box">height 100%</div>
```

结果如图 3.9 所示。

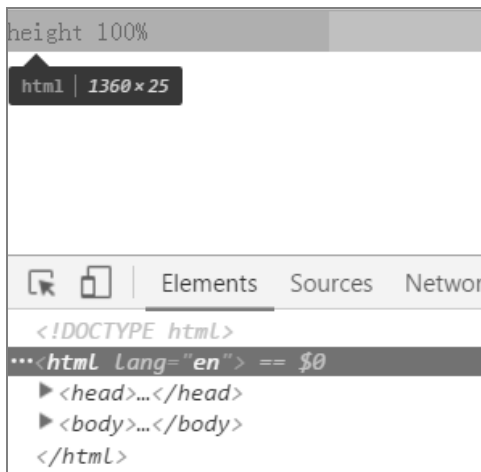


图 3.9 html 的高度为行高的高度

可以看到 html 的高度为行高的高度。也就是说，在高度为 0 或者不设置高度的情况下，高度是文字的行高。但是如果设置了高度，那么用的就是设置的高度而不再是行高了，代码如下。

```
<style>
  html,body,div{
    margin:0;
    padding:0;
  }
  div{
    width:200px;
    height:100px;
    line-height:25px;
    background-color:pink;
  }
</style>
```

结果如图 3.10 所示。



图 3.10 设置了高度则用高度

CSS 总是变化莫测的，当遇上更复杂的情况时，往往出乎预期的情况。例如下面这种情况，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    position:absolute;
    width:200px;
    height:100%;
    background-color:pink;
  }
</style>
<div>css 100%</div>
```

结果如图 3.11 所示。

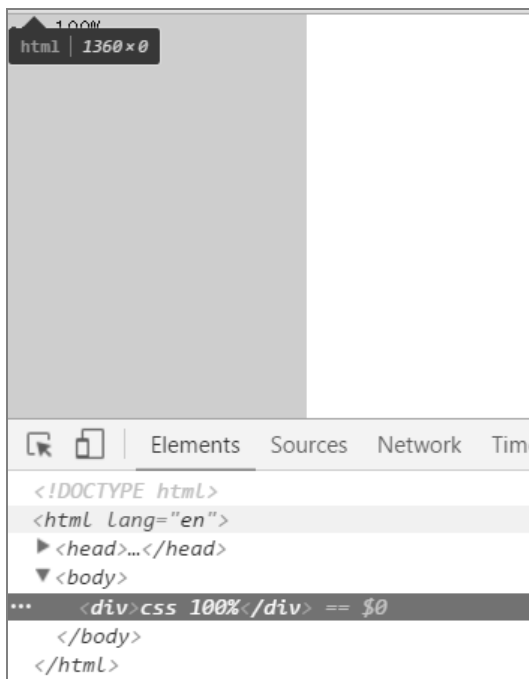


图 3.11 将高度设置为 100%

将高度设置为 100%竟然生效了，但 html 的高度并没有自适应 div 的高度，真是彻底地脱离标准流了，在第 2 章讲过，如果绝对定位没有定位的祖先元素，则包含块为初始包含块。这里的百分比是参照可视区的大小来计算的，比如设置高度为 1%，则是可视区高度总和的 1/100，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    position:absolute;
    width:100px;
    height:50%;
    background-color:red;
  }
</style>
<div></div>
```

结果如图 3.12 所示。

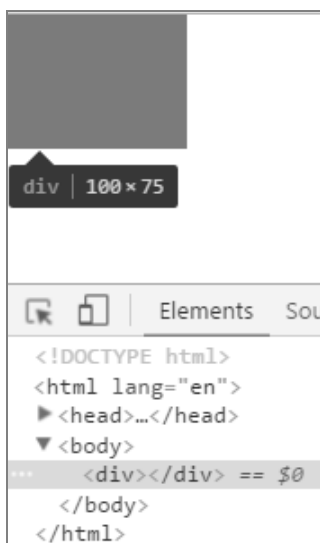


图 3.12 百分比参照可视区大小来计算

将 div 的 height 设置为 50%，它的高为 75px，1% 就是 $75/50 = 1.5$ 。如果将 height 设置为 100%，那么 div 就应该是 150。将 height 设置为 100% 的结果如图 3.13 所示。

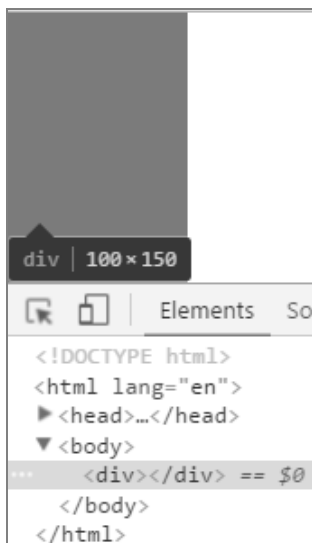


图 3.13 实验结果

代码如下。

```
div{  
    position:absolute;  
    width:100px;  
    height:100%;  
    background-color:red;  
}
```

在绝对定位中如果高度使用 100%，还需要注意，它只是一个视口的高度，如图 3.14 所示。

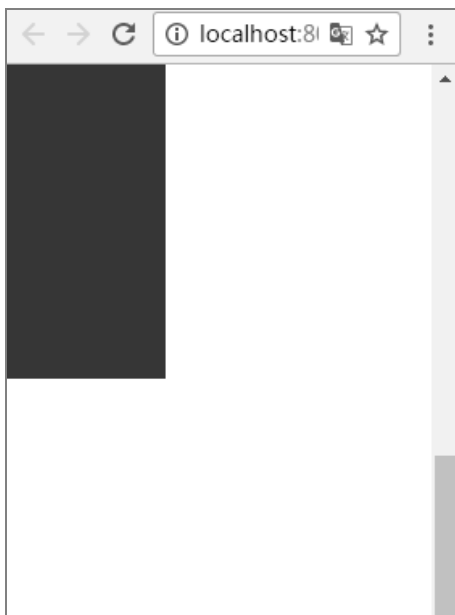


图 3.14 小问题

从图 3.14 中可以看到它的高度并没有占满全屏，代码如下。

```
<style>  
    body,div,p{  
        margin:0;  
        padding:0;  
    }  
    div{  
        position:absolute;  
        left:0;
```

```
        top:0;
        width:100px;
        height:100%;
        background-color:red;
    }
    p{
        height:1000px;
    }
</style>
<div></div>
<p></p>
```

也就是说，绝对定位参照的是一个视口的高度。如果想解决这个问题，先来看一个例子，代码如下。

```
<style>
    body,div{
        margin:0;
        padding:0;
    }
    .box{
        position:relative;
        width:200px;
        height:100px;
    }
    .box .item{
        position:absolute;
        width:100%;
        height:100%;
        background-color:pink;
    }
</style>
<div class="box">
    <div class="item">item</div>
</div>
```

效果如图 3.15 所示。

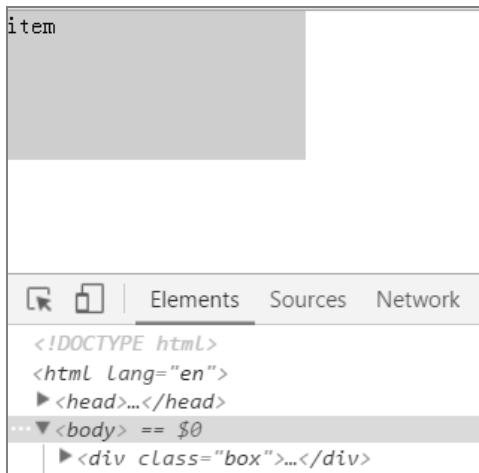


图 3.15 绝对定位百分比参照的是父元素

如果一个绝对定位的元素在另外一个定位元素里面（除 `position` 为 `static` 以外），那么其百分比参照的是父元素生成的包含块计算出来的值。

因此想让定位元素的高度占满整个屏幕，可以这样改写代码，代码如下。

```
body{  
    position:relative;  
}
```

这样它参照的就是 `body` 了，效果如图 3.16 所示。

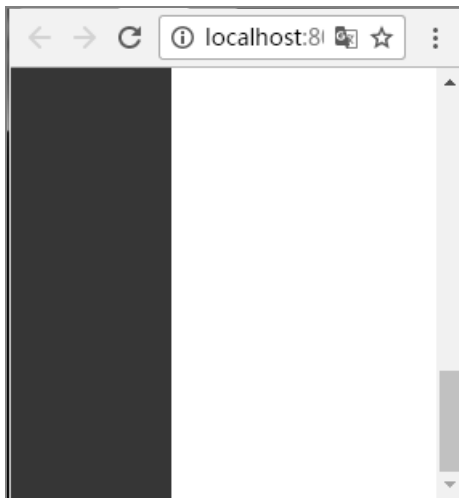


图 3.16 参照 body

虽然百分比很好用，但如果使用不当，问题也很多，比如下面的这个例子，将 `div` 元素的 `margin-left` 属性设置为 100%，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    width:100px;
    height:100px;
    margin-left:100%;
    background-color:pink;
  }
</style>
<div>CSS 100%</div>
```

效果如图 3.17 所示。

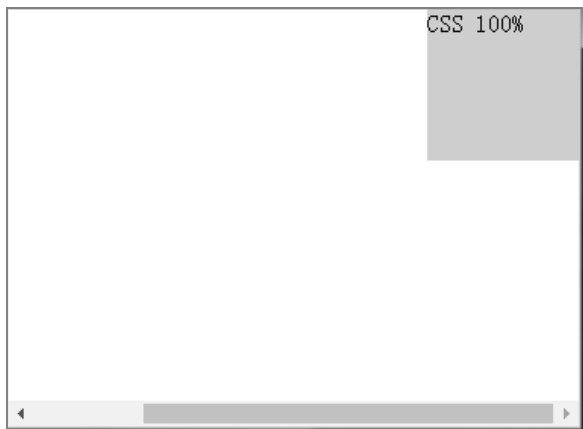


图 3.17 将 `div` 元素的 `margin-left` 属性设置为 100%

你会发现出现了滚动条，这是因为将 `div` 的 `margin-left` 设置成了 100%，而百分比是参照其包含块 `body` 的宽度，`body` 又是参照的 `html`（某些浏览器将它当作初始包含块）。而 `html` 的初始包含块是可视区，所以可视区的宽度再加上元素的 100px，自然就超出屏幕了。一般第一反应是给 `margin-right` 设置一个 -100px，但是它不是 `left` 和 `right`。虽然 CSS3 出了 `box-sizing` 属性，可以改变元素的盒模型组成模式，但却没有“`margin-box`”，所以这是行不通的。我们的目的就是为减去 100px，用一个函数就可以解决这个问题，代码如下。

```

<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    width:100px;
    height:100px;
    margin-left:calc(100% - 100px);
    background-color:pink;
  }
</style>
<div>CSS 100%</div>

```

效果如图 3.18 所示。

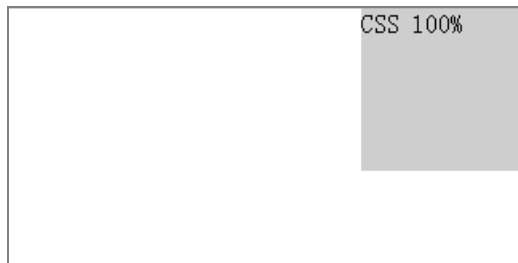


图 3.18 通过 calc 函数解决

这里使用了 calc 函数，calc 函数允许进行计算功能，但需要注意的是，前一个数和后一个数之间要有一个空格。calc 函数不但可以进行加、减、乘、除运算，还可以对%、px、em、rem、vw 等进行运算。

还可以用一个更简单的方法，让 body 的宽减去 100px，即给它的父元素增加 margin 就可以了，因为它的百分比是参照其父元素的宽度计算的，因此这里将它的父元素的宽度减小，那么它的 margin-left 的 100%自然就小了，代码如下。

```

<style>
  body,div{
    margin:0;
    padding:0;
  }
  body{

```

```

    margin-right:100px;
  }
  div{
    width:100px;
    height:100px;
    margin-left:100%;
    background-color:pink;
  }
</style>
<div>CSS 100%</div>

```

效果如图 3.19 所示。

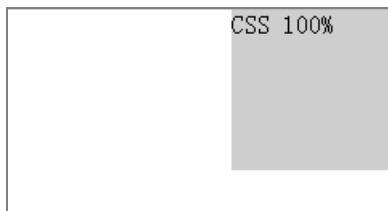


图 3.19 给 body 加 margin 解决

这样也没有滚动条了，虽然这样比较简单，但需要注意的是，如果这么做每个元素就都有 100px 的外间距了。

对百分比的一些特性总结。

- 标准流中的元素，如果其属性是可继承的，则参照的是通过继承父元素计算后的值，如果不可继承则参照初始值；
- 绝对定位参照的是离它最近的父元素或祖先元素，如果没有父元素或祖先元素，那么参照的是初始包含块（不同的浏览器可能不一样，因为 W3C 没有规定浏览器具体要如何去实现）。但实际上，大部分浏览器将可视区当作绝对定位的包含块；
- 固定定位参照可视区。

3.2 探索 auto 密码

将一个元素的 CSS 属性值设置为 auto，或不设置成 auto，它们会有什么区别呢？先看一个将 CSS 属性值设置为 auto 的例子，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    width:auto;
    height:35px;
    background-color:pink;
  }
</style>
<div class="box">auto</div>
```

效果如图 3.20 所示。



图 3.20 将 CSS 属性值设置为 auto 的效果

是否设置 auto，从表面上看好像效果一样。如果将此 div 的 margin 设置为 auto，那又会是什么样呢？代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    width:100px;
    margin:0 auto;
    background-color:pink;
  }
</style>
<div class="box">auto</div>
```

效果如图 3.21 所示。

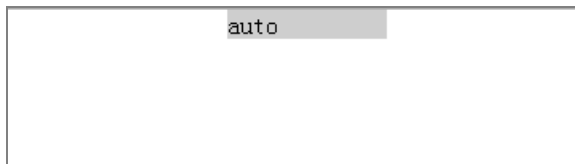


图 3.21 效果为水平居中

显示的内容水平居中了，那么为什么会水平居中呢？先将代码进行修改，代码如下。

```
div{  
    width:100px;  
    margin-left:auto;  
    background-color:pink;  
}
```

修改代码后的效果如图 3.22 所示。

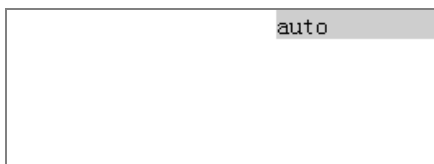


图 3.22 div 移动右上角

从图 3.22 可以看到，显示的内容跑到了右上角，那么将 `margin-right` 设置为 `auto`，结果应该很明显了，代码如下。

```
div{  
    width:100px;  
    margin-right:auto;  
    background-color:pink;  
}
```

修改后的效果如图 3.23 所示。



图 3.23 div 移动左上角

在第2章中讲过一个元素会被其包含块所包围，这里的 `auto` 是指将包含块剩余的空间占满，如图 3.24 所示。

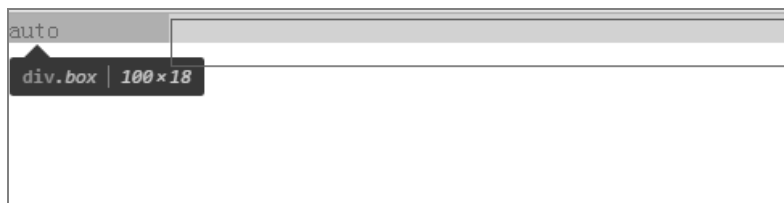


图 3.24 占满剩余空间

在图 3.24 中画线的位置（空闲空间）都被 `auto` “占领”了，所以在图 3.22 中，`div` 移到了右上角。

块级非替换元素：包含块的宽度 = `'margin-left' + 'border-left-width' + 'padding-left' + 'width' + 'padding-right' + 'border-right-width' + 'margin-right'`。

- 如果只有一个值指定为 `auto`，那么它的计算值为剩余的空间；
- 如果将宽度的值设置为 `auto`，则其他属性的 `auto` 都会变成 0，剩余的空间都会成为 `width` 的值；
- 如果将 `margin-left` 和 `margin-right` 都设置为 `auto`，则它们的计算值相等。也就是剩余的空间一人一半。

不过在有些情况下，设置 `auto` 是没有意义的，如行内元素，代码如下。

```
<style>
  span{
    margin-left:auto;
  }
</style>
<span>auto</span>
```

如果将行内元素 `margin-left` 或 `margin-right` 设置为 `auto`，其结果都是 0，另外 `width` 对于行内元素也是无效的。

但是如果将一个替换元素的 `width` 设置为 `auto`，那么其结果值为其元素的内在宽度。比如 `img` 元素，如果把它宽度设置为 `auto`，那么其结果就是图片本身的宽度。但往往没必要设置，因为它的默认值就是 `auto`。

实际上，在宽度方面设置 `auto` 没有太大意义，因为即使不设置，它们的默认值也是 `auto`。不过当定位的属性值为 `absolute` 或 `fixed` 时，如果将其宽度设置为 `auto`，情况就会有所不同，

可能这么做会比较复杂，但可以使 `auto` 在定位中大显身手，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    position:absolute;
    width:100px;
    margin-left:auto;
    background-color:pink;
  }
</style>
<div>auto</div>
```

此时 `auto` 不起效果，但当给 `left` 和 `right` 加上 0 之后 `auto` 就生效了，效果如图 3.25 所示。

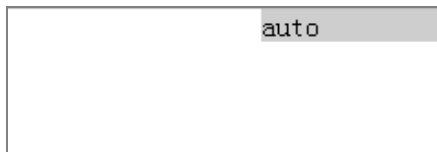


图 3.25 需要给 `left` 和 `right` 加上 0

也就是说，`left` 和 `right` 会影响 `margin` 的 `auto`，即在定位中如果想让一个元素水平居中，那么还得将对应的 `left` 和 `right` 设置为 0，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    position:absolute;
    width:100px;
    left:0;
    right:0;
    margin:0 auto;
    background-color:pink;
  }
</style>
```



```
    }  
</style>  
<div>auto</div>
```

效果如图 3.26 所示。

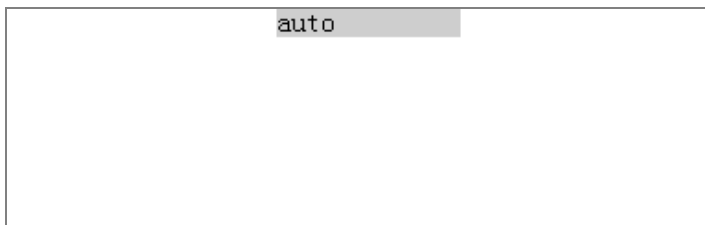


图 3.26 水平居中

既然如此，那么如果将 `top` 和 `bottom` 都设置为 0，然后将 `margin` 设置为 `auto`，是不是就成了一个垂直水平居中的样子了呢？代码如下。

```
<style>  
    body,div{  
        margin:0;  
        padding:0;  
    }  
    div{  
        position:absolute;  
        width:100px;  
        height:100px;  
        left:0;  
        right:0;  
        top:0;  
        bottom:0;  
        margin:auto;  
        background-color:pink;  
    }  
</style>  
<div>auto</div>
```

效果如图 3.27 所示。

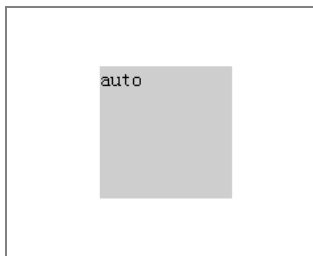


图 3.27 垂直水平居中

显示的内容果然是水平垂直居中的，这里需要注意一个问题，请仔细看代码，会发现代码中给元素加上了一个高度，代码如下。

```
div{  
    position:absolute;  
    width:100px;  
    height:100px;  
    left:0;  
    right:0;  
    top:0;  
    bottom:0;  
    margin:auto;  
    background-color:pink;  
}
```

如果把高度这行代码去掉，则效果将如图 3.28 所示。

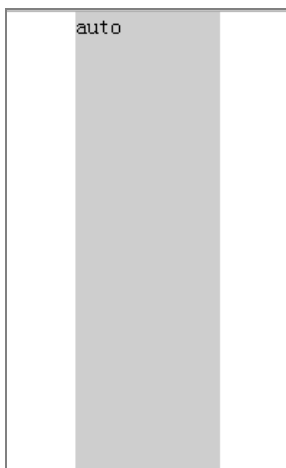


图 3.28 把高度去掉以后

如果你认为出现这种效果是受 `auto` 的影响那就错了,其实是受了 `top` 和 `bottom` 的影响。将代码进行修改,代码如下。

```
<style>
body,div{
    margin:0;
    padding:0;
}
div{
    position:absolute;
    left:0;
    right:0;
    top:0;
    bottom:0;
    background-color:pink;
}
</style>
```

效果如图 3.29 所示。

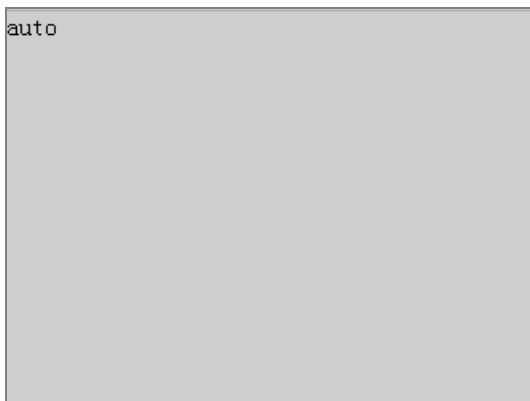


图 3.29 `top` 和 `bottom` 影响了 `auto` 的显示位置

那么现在我们就得重新思考,这个 `left` 和 `top` 究竟都做了些什么? 下面来看几个例子,代码如下。

```
<style>
body,div{
    margin:0;
    padding:0;
```

```
    }  
    div{  
        position:absolute;  
        left:0px;  
        right:0px;  
        background-color:pink;  
    }  
</style>  
<div>auto</div>
```

效果如图 3.30 所示。



图 3.30 演示

再来看一段代码。

```
<style>  
    body,div{  
        margin:0;  
        padding:0;  
    }  
    div{  
        position:absolute;  
        left:10px;  
        right:0px;  
        background-color:pink;  
    }  
</style>  
<div>auto</div>
```

效果如图 3.31 所示。



图 3.31 给 left 设置偏移量

将 `left` 设置为 `10px` 就出现了图 3.31 显示的效果。实际上 `left` 和 `right` 是偏移量，如果想更简单地理解，可以假设这个 `left` 就是起点，而 `right` 作为一个终点，元素的宽度就由这两个值决定。`top` 和 `bottom` 作为垂直方向的计算，`top` 为起点，`bottom` 为终点。在默认情况下，`left`、`top`、`bottom`、`right` 都为 `auto`。

绝对定位元素：包含块的宽 = `'left' + 'margin-left' + 'border-left-width' + 'padding-left' + 'width' + 'padding-right' + 'border-right-width' + 'margin-right' + 'right'`

为了更好地理解，重新看一下之前讲的一个例子，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    width:100px;
    margin-left:auto;
    background-color:pink;
  }
</style>
<div>auto</div>
```

效果如图 3.32 所示。



图 3.32 div 移到右上角

为什么显示的内容会移到右上角呢？大家都知道 `div` 是一个块元素，默认是占一行的，但给它设置了宽度，所以它只有 `100px` 的宽度，那么剩下的空间就不属于 `width` 了。而当设置 `margin-left` 为 `auto` 时，`div` 就移到右上角了，因为这个 `margin-left` 把 `div` 剩下的空间都占了。

当将一个行内或者行内块设置 `margin-left` 为 `auto` 时是没有效果的，代码如下。

```
<style>
  body,div{
```

```
margin:0;
padding:0;
}
div{
display:inline-block;
width:100px;
margin-left:auto;
background-color:pink;
}
</style>
<div>auto</div>
```

效果如图 3.33 所示。



图 3.33 行内和行内块设置 margin 无效

因为行内和行内块的空间只有自身的那些位置，而那些位置又都让宽度和高度占了，所以在行内和行内块中设置 auto 就没有效果了。对于宽度和高度来说，设置 auto 始终是没有意义的，因为它是根据内容计算的。这也可以用来解释为什么说定位了的元素（position 为 absolute、fixed 时）要设置 left 和 right 都为 0，“margin:0 auto”才会起效果。因为定位的元素和行内块一样，宽度和高度默认是由内容决定的，而这个宽度和高度都被内容占了，所以就没有效果了。而当设置 left 和 right 为 0 时，它将剩余的空间扩大。就像块级元素一样，虽然看不见，但如果设置了它的宽度和高度，剩下的空间就由其他属性分配。

对于浮动元素来说，如果设置 width、margin-left、margin-right 为 auto，那么其结果都为 0，宽度由内容来决定，代码如下。

```
<style>
body,div{
margin:0;
padding:0;
}
div{
float:left;
width:auto;
height:100px;
```

```
background-color:red;
}
</style>
<div>auto</div>
```

对于浮动的替换元素来说，如果将 width 设置为 auto，依然是元素自身的宽度。

总结

水平方向的情况：

- 如果将一个替换元素的宽度设置为 auto，则宽度等于内在宽度；
- 对于行内元素来说，width 并不适用，其他的几个属性，比如当 margin-left 设置为 auto 时，结果也都是 0；
- 块类元素将属性设置为 auto 时，则占满剩余空间；
- 对于绝对定位元素，如果只有一个属性的值为 auto，则占满剩余的空间。

垂直方向的情况：

- 将替换元素设置 auto，其结果为 0。如果将一个替换元素的高度设置为 auto，则由内在高度决定；
- 将块类、非替换元素、浮动的非替换元素设置 auto，其结果为 0，如果是 height，则取决于子元素高的总和（不包括浮动元素和绝对定位，相对定位只考虑未偏移的情况）；
- 绝对定位如果只有一个属性的值为 auto，则占满剩余空间。

3.3 设计响应式网页 rem

rem 是设计响应式网页的“神器”，因为 rem 单位是相对于 html 元素的 font-size 属性的，因此当使用 rem 作为属性单位时，当改变 html 元素的 font-size，其他使用 rem 作为单位的元素也会跟着适配大小，代码如下。

```
<style>
body,div{
margin:0;
padding:0;
}
```

```
html{
  font-size:20px;
}
.box{
  width:10rem;
  height:10rem;
  font-size:2rem;
  background-color:pink;
}
</style>
<div class="box">rem</div>
```

这里给 html 设置一个 20px 的字体大小,对应的.box 使用了 rem,效果如图 3.34 所示。



图 3.34 相对单位 rem

可以看到它是一个 200×200 的盒子,为什么是 200×200 呢?看看它们的换算。

当 html 的 font-size = 10px;时, 1rem = 10px; 2rem = 20px; 3rem = 30px;

当 html 的 font-size = 15px;时, 1rem = 15px; 2rem = 30px; 2.2rem = 33px;

知道了它们之间是如何换算的,就可以根据屏幕的大小制作一个响应式的网页。那么怎么知道用户的屏幕大小呢?有以下 3 种方法可以解决这个问题。

(1) 用 JavaScript 来获取屏幕大小,动态设置 html 字体大小。该方法的优点是兼容性

强；缺点是步骤烦琐，而且还会出现页面卡顿的情况；

(2) 用 CSS 媒体查询。该方法的优点是操作简单；缺点是只有在临界点的时候才会触发，兼容性一般；

(3) 用相当于视口宽度单位的 vw 值。该方法的优点是操作简单，而且精确；缺点是兼容性差。

这里先用 CSS 媒体查询的方法来做讲解，所谓的 CSS 媒体查询就是可以根据一些条件来写特定的规则，其中包括了屏幕查询，也就是可以指定一个屏幕大小，当屏幕是你指定的大小时，它就会运行这里面的代码。

下面看一段代码，代码如下。

```
@media screen and (min-width: 320px) {  
  html {  
    font-size: 16px;  
  }  
}
```

此段代码的意思是，当屏幕的宽度不小于 320px 时，就会运行这段代码，否则没有效果。

@media 可以针对不同的屏幕尺寸设置不同的样式，当重置浏览器大小时，页面也会根据浏览器的宽度和高度重新渲染。其中 screen 主要用来查询电脑、平板电脑、智能手机等屏幕大小；and 就是并且的意思，这整句话的意思就是查询设备的屏幕宽度是否小于 320px。这里用了一个 min-width，min-width 表示最小宽度。如果屏幕的宽度小于 320px，那么就不会应用此规则，因为设置的是屏幕最小宽度为 320px。

所以要制作一个自适应的网页，代码如下。

```
<style>  
  body,div{  
    margin:0;  
    padding:0;  
  }  
  html{  
    font-size:12px;  
  }  
  @media screen and (min-width: 320px) {  
    html {  
      font-size: 14px;  
    }  
  }
```

```
    }  
    @media screen and (min-width: 640px) {  
        html {  
            font-size: 16px;  
        }  
    }  
    @media screen and (min-width: 1000px) {  
        html {  
            font-size: 18px;  
        }  
    }  
    .box{  
        width:10rem;  
        height:10rem;  
        font-size:2rem;  
        background-color:pink;  
    }  
</style>  
<div class="box">rem</div>
```

在不同的屏幕下，对应的 div 大小也会改变。了解更多关于 CSS 媒体查询方面的资料：
https://developer.mozilla.org/zh-CN/docs/Web/Guide/CSS/Media_queries。

目前大部分的移动端网站都是使用这种方式布局，但它还是有缺陷的，比如下面这段代码。

```
@media screen and (min-width: 320px) {  
    html {  
        font-size: 14px;  
    }  
}  
@media screen and (min-width: 640px) {  
    html {  
        font-size: 16px;  
    }  
}
```

这段代码的意思是只有当屏幕宽度不小于 320px，并且不大于 640px 时，才会运行 14px。当屏幕宽度最小为 640px 时，就会运行 16px。要表明的意思是，它并不能控制更加精确的尺寸。除非挨个去写规则，但是这样做相当烦琐。

3.4 vw、vh、vmin、vmax 基于视口单位

vw、vh、vmin、vmax 是新增的单位，这些单位是相对于视口的，也就是可视区域，总共分成了 100 份。如果想要一个元素宽度为可视区域的 50%，那么可以用 50vw 来表示。在大多数情况下百分比是根据继承的父元素的计算值来做参照的，所以完全与 vw 单位没法比。

- vw 可视区域宽度，总宽度为 100vw；
- vh 可视区域高度，总高度为 100vh；
- vmin 比较可视区域宽度和高度，哪个更小就用哪个；
- vmax 比较可视区域宽度和高度，哪个更大就用哪个。

先举个例子，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:100vw;
    height:30vh;
    font-size:10vw;
    background-color:pink;
  }
</style>
<div class="box">vw</div>
```

使用这些单位后，它会自动响应视口宽度。当宽度改变时，这个元素的大小也会改变。因为这一点，它才有了价值，可以通过它来实现自适应网页布局（不过还得配合 rem 单位才行）。

如果想让一个字体的大小在指定的区间内变化，比如可视区域的大小在 980px~320px 时，让字体的大小在 14~20 之间变化，那么可以这样。

$$14 + (20 - 14) \times (980 - 320) / (980 - 320)$$

$$\text{简化后为: } 14 + 6 \times (980 - 320) / 660$$

动态: $14\text{px} + 6 \times (100\text{vw} - 320\text{px}) / 660$

这样字体大小就在 14~20 之间变化了。

如果对举的这个例子还不明白, 则可以这样计算, 假设字体的大小要在 14~20 之间, 那么 14 是必须设置的, 所以肯定得加 14px, 后面根据屏幕可视区域的大小来计算。

那么代码如下。

```
html{
  font-size:14px + 6 * (100vw - 320px) / 660;
}
```

不过不能直接这么使用, 还得用 calc 函数, 因此代码如下。

```
html{
  font-size:calc(14px + 6 * (100vw - 320px) / 660);
}
```

如果屏幕宽度大于 980px, 那么字体的大小就不是 14~20 了。所以, 如果要做更加精确的处理, 可以结合 CSS 媒体查询, 代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  html{
    font-size:20px;
  }
  @media screen and (max-width:980px){
    html{
      font-size:calc(14px + 6 * (100vw - 320px) / 660);
    }
  }
  .box{
    width:10rem;
    height:10rem;
    font-size:2rem;
    background-color:pink;
  }
</style>
<div class="box">rem</div>
```

当屏幕宽度小于 980px 时，效果如图 3.35 所示。当屏幕宽度大于 980px 时，效果如图 3.36 所示。

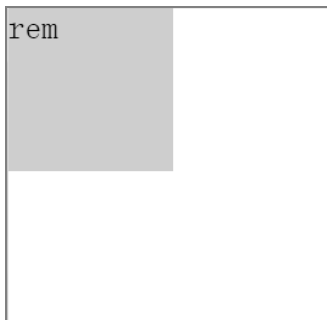


图 3.35 当屏幕宽度小于 980px 时

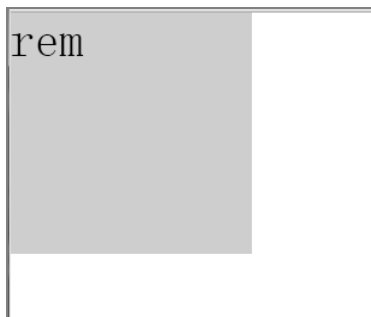


图 3.36 当屏幕宽度大于 980px 时

这里用了 `max-width` 属性，这个属性的意思是宽度最大只能是 980px，如果屏幕宽度大于 980px 就不起作用了。它会根据屏幕的大小而响应，不过兼容性还有待提高。因为放不了动态图片，所以最好自己动手试试浏览器是否支持。

如果某个值可以继承，则百分比参照的是父元素计算的值。而当父元素不设置高度时，则是根据子元素高度来确定的。所以，如果没有直接设置宽度和高度，100%的设置是没有作用的，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:100%;
    height:100%;
    background-color:pink;
  }
</style>
<div class="box"></div>
```

如果用视口单位，就绝对不会出现这种情况，因为它是相对屏幕可视区的，代码如下。

```
.box{
  width:100vw;
  height:100vh;
```

```
background-color:pink;  
}
```

如果要制作水平垂直居中显示的内容，则代码如下。

```
<style>  
body,div{  
    margin:0;  
    padding:0;  
}  
.box{  
    width:100px;  
    height:100px;  
    margin-top:calc(50vh - 50px);  
    margin-left:calc(50vw - 50px);  
    background-color:pink;  
}  
</style>  
<div class="box">hello css</div>
```

效果如图 3.37 所示。

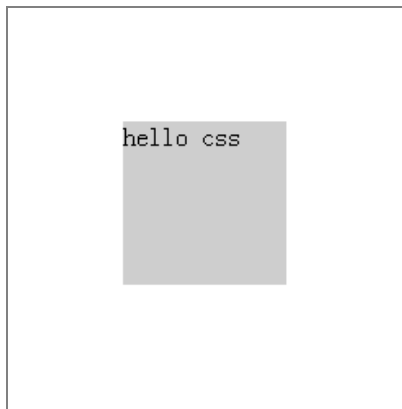


图 3.37 水平垂直居中

可以像定位一样很简单地实现这个功能。用视口单位实现网格布局效果也非常不错，如图 3.38 所示。

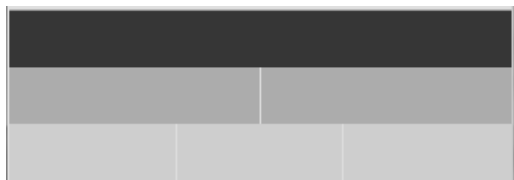


图 3.38 用视口单位实现网格布局

代码如下。

```
<style>
    body,div{
        margin:0;
    }

    .column-1{
        float:left;
        width:100vw;
    }
    .column-2{
        float:left;
        width:calc(100vw / 2);
    }
    .column-3{
        float:left;
        width:calc(100vw / 3);
    }

    body > div{
        overflow:hidden;
    }
    div > div{
        height:35px;
        outline:1px solid #dedede;
    }
    .box-1 div{
        background-color:red;
    }
    .box-2 div{
        background-color:orange;
    }
```

```

    }
    .box-3 div{
        background-color:pink;
    }
</style>
<div class="box-1">
    <div class="column-1"></div>
</div>
<div class="box-2">
    <div class="column-2"></div>
    <div class="column-2"></div>
</div>
<div class="box-3">
    <div class="column-3"></div>
    <div class="column-3"></div>
    <div class="column-3"></div>
</div>

```

另外，当需要将图片按照屏幕的比例显示时，用视口单位也是不错的选择。

3.5 什么是 ch

ch 是一个相对于数字 0 的大小。比如定义了 5ch 的宽度，那么就只能装下 5 个 0，代码如下，效果如图 3.39 所示。

```

<style>
    body,div{
        margin:0;
        padding:0;
    }
    .box{
        width:5ch;
        background-color:pink;
    }
</style>
<div class="box">000000</div>

```

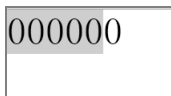



图 3.39 5ch 的宽度效果

如图 3.39 所示，多出的最后一个 0 溢出了。其实它是有规律的，即 1ch = 1 个英文 = 1 个数字，2ch = 1 个中文，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div:first-of-type{
    width:3ch;
    font-size:50px;
    background-color:pink;
  }
  div + div{
    width:16ch;
    font-size:50px;
    background-color:red;
  }
</style>
<div>CSS</div>
<div>全解探索未知的迷</div>
```

效果如图 3.40 所示。

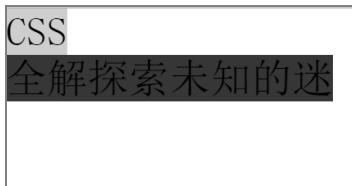


图 3.40 2ch = 1 个中文

如果项目需要限制输入个数，则可以使用以下代码。

```
<style>
  body,div{
    margin:0;
```

```
padding:0;
}
h1{
width:18ch;
overflow:hidden;
white-space: nowrap;
text-overflow: ellipsis;
font-size:50px;
background-color:pink;
}
</style>
<h1>标题被限制输入了，超出隐藏噢。</h1>
```

效果如图 3.41 所示。

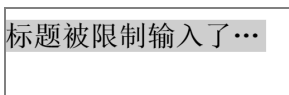


图 3.41 限制输入个数

这里使用了以下代码。

```
overflow:hidden;
white-space: nowrap;
text-overflow: ellipsis;
```

此段代码的意思是超出的部分显示省略号；white-space 是防止换行；text-overflow 是省略号；overflow:hidden 是超出隐藏。

不过这个方法是有缺陷的，如果某段文字是中文加英文混合的内容，那么这个设置就显得不近人情了。如果想解决这个问题，就得借助 JavaScript 解决。

3.6 min、max 的巧妙运用

在 CSS 中还有两个表示宽度的属性，即 min-width 和 max-width；两个表示高度的属性，即 min-height 和 max-height。先举个例子，代码如下。

```
<style>
.box{
min-width:300px;
```

```

    height:100px;
    outline:1px solid #dedede;
}
</style>
<div class="box">hello minwidth</div>

```

当屏幕宽度大于 300px 时，效果如图 3.42 所示。当屏幕宽度大于 300px 时，会出现横向滚动条；当屏幕宽度小于 300px 时，并不会出现滚动条，效果如图 3.43 所示。

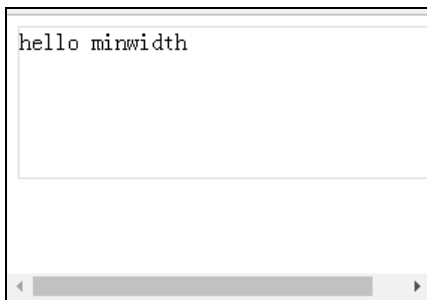


图 3.42 当屏幕宽度大于 300px 时



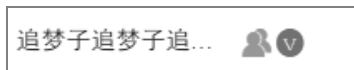
图 3.43 当屏幕宽度小于 300px 时

也就是说，一个盒子一旦设置了 `min-width` 这个属性，那么它的宽度至少是 `min-width` 中设置的那个值。`max-width` 同理，最大只能是这么大。它们的使用场景非常广泛，效果如图 3.44 所示。

图 3.44 `max-width`

这个效果是希望这几个图标一直保持在它的后面，但间距不要太大。需求看起来很简单，使用 “`display:inline-block;`” 就可以了，但还有一个需求，就是如果图中网名达到一定长度就让它隐藏起来，效果如图 3.45 所示。

而使用隐藏属性就又面临一个问题，就是它必须设置元素的宽度。但如果使用 `width` 来设置，那么在文字比较少数的情况下，文字和图标会有很大的间距，效果如图 3.46 所示。

图 3.45 `inline-block`图 3.46 `width`

若使用 `max-width` 设置，效果如图 3.47 所示。

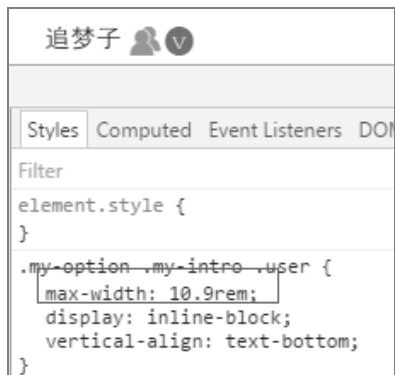


图 3.47 max-width

可以看到，当使用 `max-width` 时，文字和图标有很大间距的问题就解决了。

在 CSS 媒体查询里也会经常使用 `max-width`。如果只想在屏幕大于或等于 320px 时运行一段代码，代码如下。

```
@media screen and (min-width:320px){
    ....
}
```

3.7 一个 none 引出的大学问

我们经常因为兼容性而给 `button`、`input` 等设置 “`border:none;`”。那么 `NONE` 和 `0` 有什么区别呢？答案是肯定的，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  button:first-of-type{
    border:0;
  }
  button:last-of-type{
    border:none;
  }
</style>
```

```
<button>0</button>
<button>NONE</button>
```

效果如图 3.48 所示。

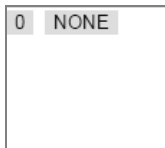


图 3.48 在 Chrome 浏览器中没问题

现在看好像 NONE 和 0 没有什么区别，但如果是在 IE 浏览器中打开，效果就不一样了，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .btn1{
    border:0;
  }
  .btn2{
    border:none;
  }
</style>
<button class="btn1">0</button>
<button class="btn2">NONE</button>
```

因为 IE 浏览器的低版本不支持伪类选择器，所以写了两个 class，效果如图 3.49 所示。

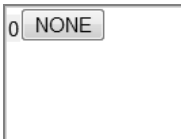


图 3.49 使用 IE 浏览器打开的效果

可以看到使用了 0 的按钮没有边框，而使用了 NONE 的按钮还有边框。也就是说，使用 NONE 相当于不渲染，而使用 0 还是会渲染的。对于性能来说肯定也是不一样的，所以尽量使用 NONE 关键字，而不要使用 0。



第 4 章 那些年我们一起定位过的元素

本章将会详细介绍定位的相关知识，以及定位的 bug 和技巧。CSS 有 3 种基本的定位机制，即普通流、浮动和绝对定位。除非专门指定，否则所有元素都在普通流中定位。也就是说，普通流中的元素的位置由元素在 HTML 中的位置决定。

4.1 定位的特点

以下是常用的几种定位属性。

- **static**: 常规框，一般使用默认值即可；
- **relative**: 在元素没有定位之前的位置上进行定位，并且它原本所占的空间仍会保留；
- **absolute**: 在没有设置 `left`、`top`、`bottom`、`right` 时，它将在没有定位之前的位置上脱离标准流，不占据空间，后面的元素将会替代它的位置，并且有层级的概念。它总是在标准流的上面，除非手动设置 `z-index`。如果设置了 `left`、`top`、`bottom`、`right`，它将会按照设置进行定位，如果它不是在一个定位的父元素里面（除 `static` 外），那么它的起点将在窗口的左上角，反之它的起点将在父元素的位置上进行定位。一个行内元素一旦设置了这个属性，其宽度、高度等属性将会生效；
- **fixed**: `fixed` 的使用效果和 `absolute` 有很多相似的地方，但也有所区别。它是按照窗口左上角定位的，除非它的父元素也是一个定位的元素（除 `static` 外），并且 `fixed` 没有设置 `left`、`bottom`、`right`、`top` 值。另外一点就是，使用了 `fixed` 的元素就像被固定住了一样，它不会随着滚动条的滚动而滚动。一个行内元素一旦设置了这个属性，其宽度、高度等属性将会生效。

当一个元素使用了除位置设置为 `static` 以外的其他几个属性，将可以配合使用 `z-index`、`left`、`right`、`top`、`bottom`，反之是没有效果的。

4.1.1 定位之 absolute 篇

通过定位可以将一部分内容与另一部分重叠，如图 4.1 所示。

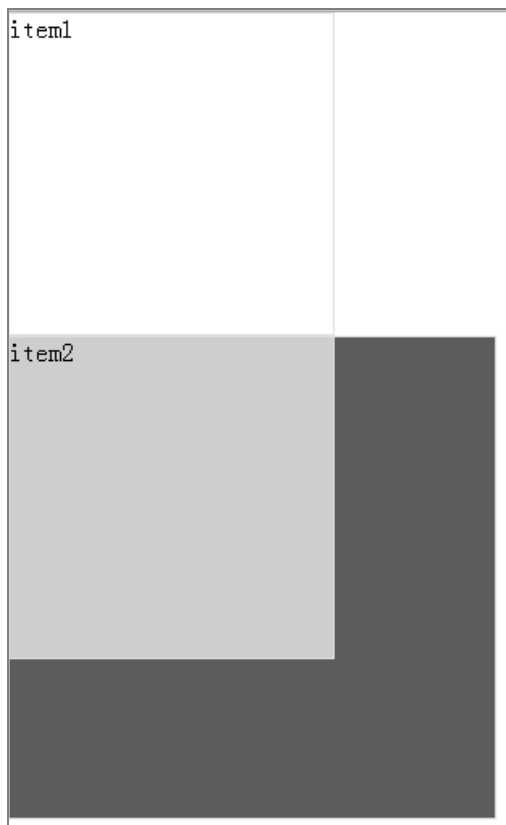


图 4.1 被重叠

代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
```

```
div{
  width:200px;
  height:200px;
  border:1px solid #dedede;
}
.item2{
  position:absolute;
  background-color:pink;
}
.item3{
  width:300px;
  height:300px;
  background-color:green;
}
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>
<div class="item3">item3</div>
```

这里给.item2 添加了 `position:absolute`，但并没有设置偏移量。在这种情况下，元素并不会改变位置（还在原地），但是却脱离了标准流，并且它将拥有层级概念，定位的元素总是在普通流的上面，除非手动指定（`z-index`）。

如果需要改变层级可以通过 `z-index`，代码如下。

```
.item2{
  position:absolute;
  z-index:-1;
  background-color:pink;
}
```

当.item2 的 `z-index` 值小于 0 时，将会被标准流盖住，如图 4.2 所示。

如果两个元素同时设置为“`position:absolute;`”，则后来者居上，按照元素的位置，越后面的层级越高，代码如下。

```
.item2{
  position:absolute;
  background-color:pink;
}
.item3{
  position:absolute;
```



```
width:300px;  
height:300px;  
background-color:green;  
}
```

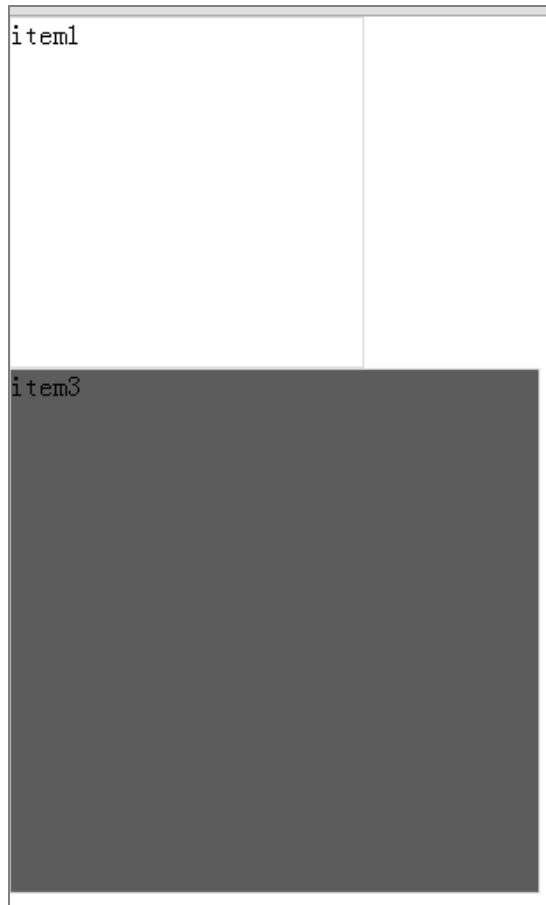


图 4.2 当 z-index 值小于 0 时将会被标准流盖住

只写 left 不写 top，代码如下。

```
.item2{  
  position:absolute;  
  left:0;  
  background-color:pink;  
}
```

不写 `top`，距离顶部也还是根据元素的当前位置。反之，只写 `top` 不写 `left` 与只写 `left` 类似，如图 4.3 所示。

同时写 `top` 和 `left`，代码如下。

```
.item2{  
    position:absolute;  
    top:0;  
    left:0;  
    background-color:pink;  
}
```

可以看到元素根据 `top` 和 `left` 的值定位，如图 4.4 所示。

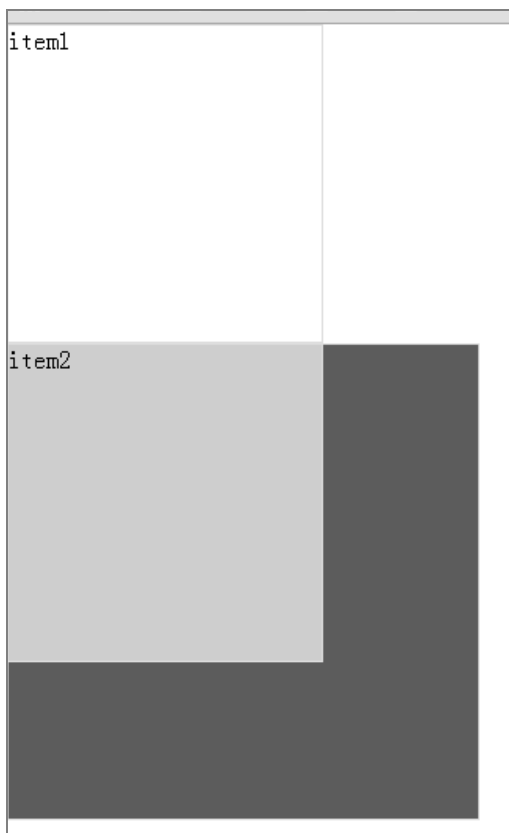


图 4.3 不写 `top` 的效果

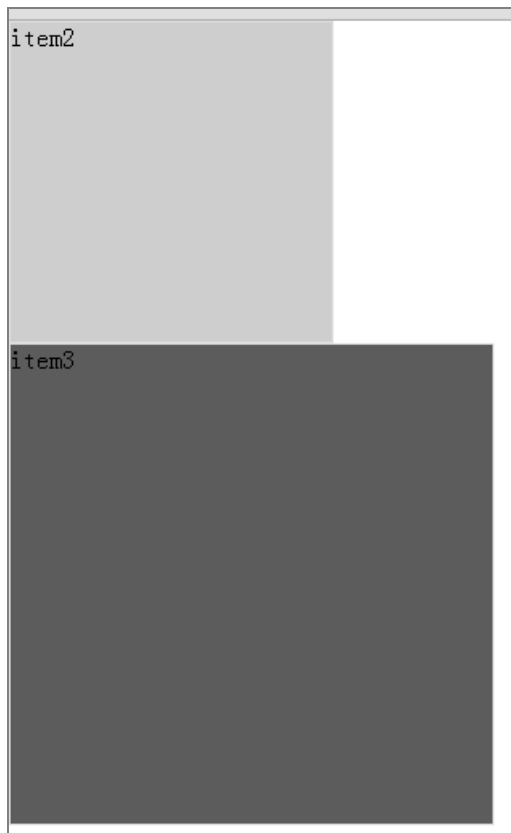


图 4.4 根据 `top` 和 `left` 的值定位

另外，如果值为 0 可以不写单位，反之报错，如图 4.5 所示。

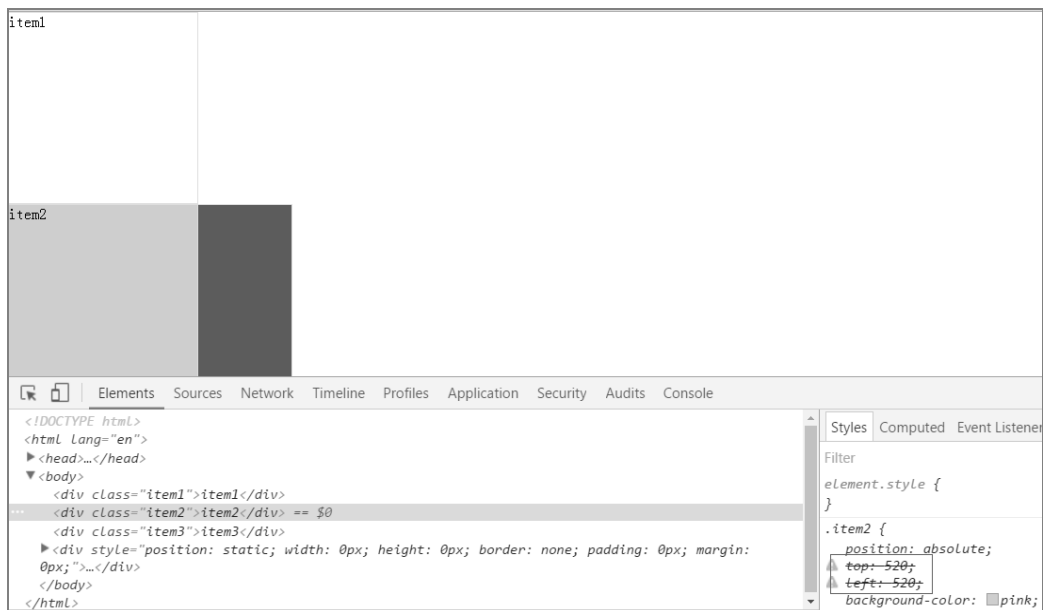


图 4.5 如果值为 0 可以不写单位

4.1.2 定位之 relative 篇

元素按照原来的位置定位，并且不会脱离标准流，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    width:200px;
    height:200px;
    border:1px solid #dedede;
  }
  .item2{
    position:relative;
    background-color:pink;
  }
</style>
<div class="item1">item1</div>
```

```
<div class="item2">item2</div>
<div class="item3">item3</div>
```

效果如图 4.6 所示。

一旦使用了此属性，它的层级就总是比标准流元素高，代码如下。效果如图 4.7 所示。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    width:200px;
    height:200px;
    border:1px solid #dedede;
  }
  .item1{
    position:relative;
    top:200px;
    left:0;
    background-color:pink;
  }
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>
<div class="item3">item3</div>
```

这里可以看到.item2 在.item1 后面，但是通过 position:relative 以后被.item1 盖住了。如果不想被.item1 盖住，也可以通过 z-index 来改变它的层级，代码如下。

```
.item1{
  position:relative;
  top:200px;
  left:0;
  z-index:-1;
  background-color:pink;
}
.item2{
  background-color:red;
}
```

效果如图 4.8 所示。

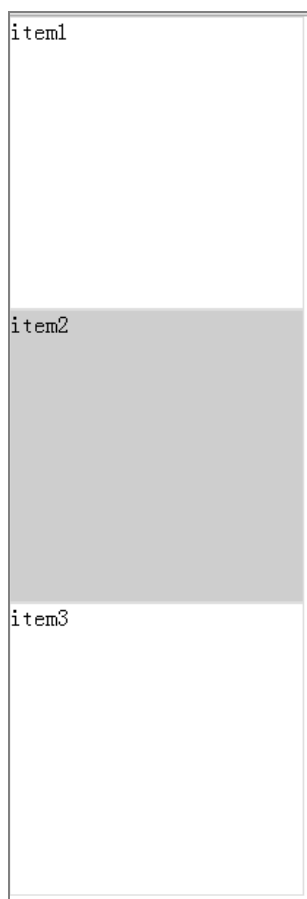


图 4.6 relative 不会脱离标准流

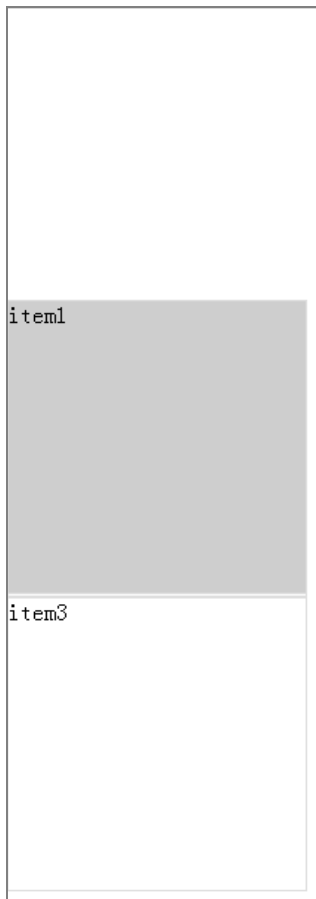


图 4.7 relative 比标准流层级高

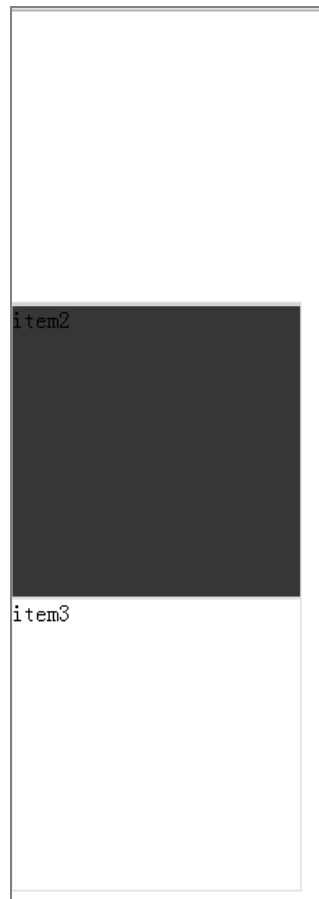


图 4.8 通过 z-index 改变层级

在默认情况下，文字会透上来，所以这里给 item2 加了一个背景颜色。

relative 的 left、right、bottom、top 都可能会引起溢出，代码如下。

```
<style>
div{
  width:100px;
  height:100px;
  border:1px solid red;
  overflow:auto;
}
```

```
p{
    position:relative;
    right:-100px;
}
</style>
<div>
    <p>relative</p>
</div>
```

效果如图 4.9 所示。

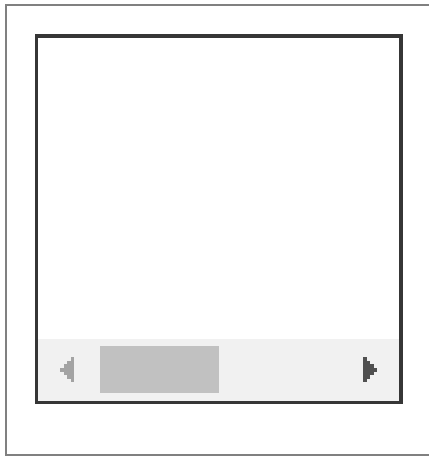


图 4.9 relative 引起溢出

导致溢出的原因是 **relative** 并不脱离标准流。

父元素定位到哪，子元素也会跟随到哪，代码如下。

```
<style>
    body,div{
        margin:0;
        padding:0;
    }
    .box{
        position:absolute;
        left:100px;
        top:100px;
        width:200px;
        height:200px;
    }
```

```
border:1px solid #dedede;
}
.item{
width:100px;
height:100px;
background-color:pink;
}
</style>
<div class="box">
  <div class="item"></div>
</div>
```

效果如图 4.10 所示。

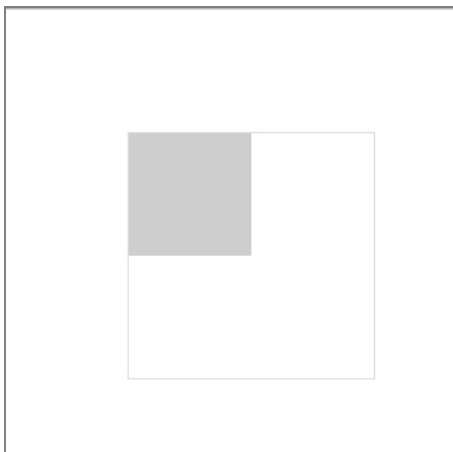


图 4.10 父元素定位到哪，子元素也会跟随到哪

4.1.3 当定位遇到定位

当子元素定位碰到父元素定位（定位嵌套定位），代码如下。

```
.box{
position:absolute;
left:100px;
top:100px;
width:200px;
height:200px;
border:1px solid #dedede;
```

```
}  
.item{  
  position:absolute;  
  left:10px;  
  top:10px;  
  width:100px;  
  height:100px;  
  background-color:pink;  
}
```

此时子元素.item 的 left 和 top 的初始值化为父元素的当前位置（也就是根据父元素的当前位置来定位），更专业地说就是根据包含块的位置来初始化位置，效果如图 4.11 所示。

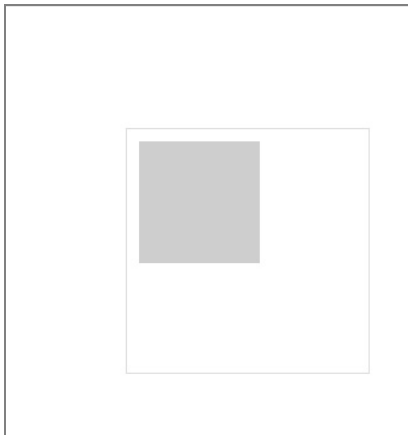


图 4.11 当子元素定位碰到父元素定位时

父元素为 position:relative;也一样，代码如下。

```
.box{  
  position:relative;  
  left:100px;  
  top:100px;  
  width:200px;  
  height:200px;  
  border:1px solid #dedede;  
}  
.item{  
  position:absolute;  
  left:10px;
```



```
top:10px;  
width:100px;  
height:100px;  
background-color:pink;  
}
```

效果如图 4.12 所示。

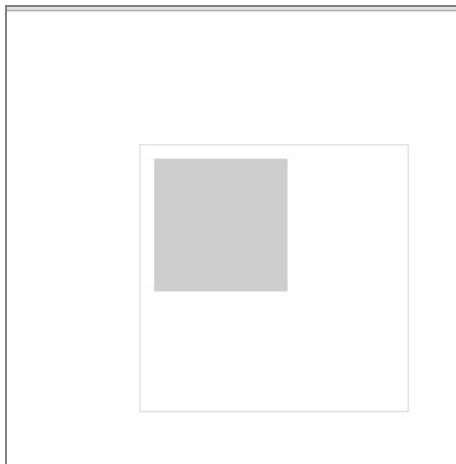


图 4.12 当父元素定位值为 relative 时

如果.item 里面再嵌套一个定位的子元素,那么这个子元素根据它的父元素.item 位置来定位,效果如图 4.13 所示。

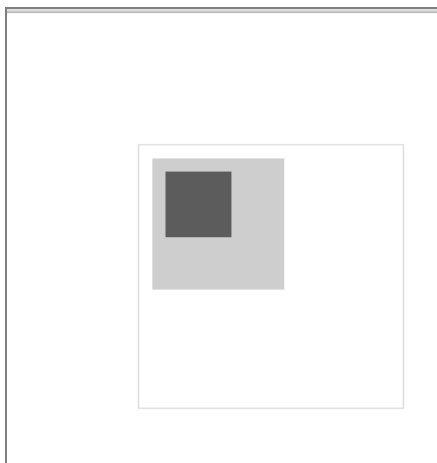


图 4.13 多层嵌套定位

代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    position:relative;
    left:100px;
    top:100px;
    width:200px;
    height:200px;
    border:1px solid #dedede;
  }
  .item{
    position:absolute;
    left:10px;
    top:10px;
    width:100px;
    height:100px;
    background-color:pink;
  }
  .itemchild{
    position:absolute;
    left:10px;
    top:10px;
    width:50px;
    height:50px;
    background-color:green;
  }
</style>
<div class="box">
  <div class="item">
    <div class="itemchild"></div>
  </div>
</div>
```

4.1.4 定位之 fixed 篇

fixed 和 absolute 的效果非常像，都会脱离标准流，但却有着天差地别。如果它在一个定位的父元素中，那么就很容易看出它们的不同之处，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box1,.box2{
    position:relative;
    left:150px;
    top:150px;
    width:200px;
    height:200px;
    border:1px solid #dedede;
  }
  .box1 .item{
    position:absolute;
    top:0;
    left:0;
    width:100px;
    height:100px;
    background-color:pink;
  }
  .box2 .item{
    position:fixed;
    top:0;
    left:0;
    width:100px;
    height:100px;
    background-color:green;
  }
</style>
<div class="box1">
  <div class="item">absolute</div>
</div>
```

```
<div class="box2">  
  <div class="item">fixed</div>  
</div>
```

效果如图 4.14 所示。

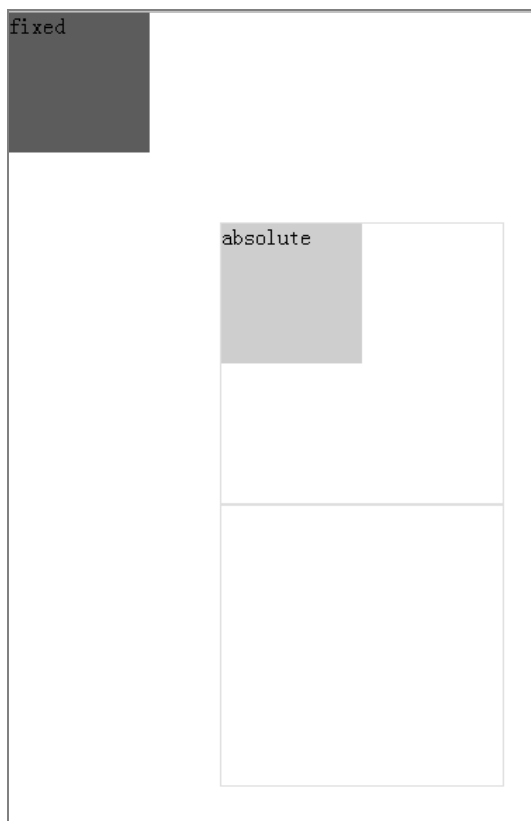


图 4.14 fixed

可以看到 `position` 为 `fixed` 的元素，并没有根据它父元素的位置进行定位，而是跑到了浏览器左上角。这就说明 `fixed` 的包含块是视口，与 `absolute` 没有定位的父元素或祖先元素的情况很像。

4.1.5 偶遇定位 bug，才知定位的真理

在给一个元素运用了定位属性的 `fixed` 值后，如图 4.15 所示。

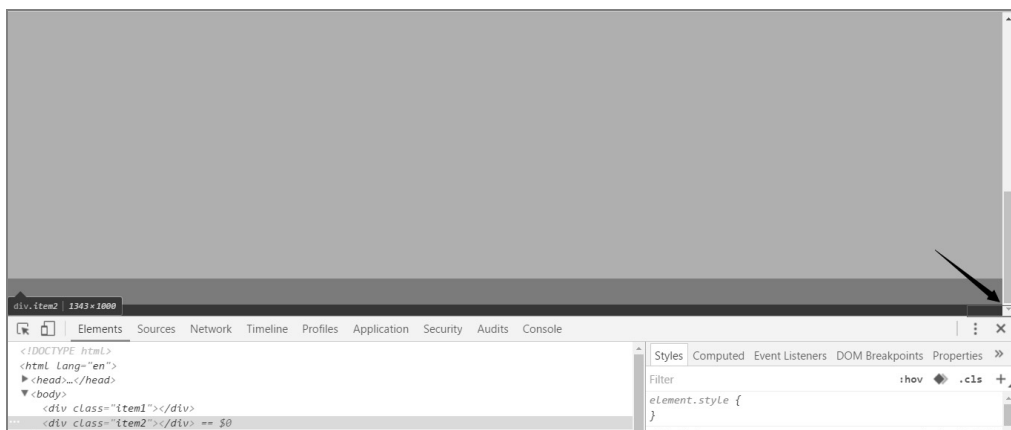


图 4.15 效果图

代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .item1{
    position:fixed;
    left:0;
    bottom:0;
    width:100%;
    height:50px;
    background-color:red;
  }
  .item2{
    width:100%;
    height:1000px;
    background-color:pink;
  }
</style>
<div class="item1"></div>
<div class="item2"></div>
```

使用 position 为 fixed 的.item1 元素把.item2 最后一部分盖住了，并且滚动不上去。另外，可以看到最后还留了一点空白。用 Chrome 浏览器测量了一下，.item2 元素大概会被盖

住 35px。因此，只需要用 “margin-bottom:35px;” 就可以解决这个问题了，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .item1{
    position:fixed;
    left:0;
    bottom:0;
    width:100%;
    height:50px;
    background-color:red;
  }
  .item2{
    width:100%;
    height:1000px;
    margin-bottom:35px;
    background-color:pink;
  }
</style>
<div class="item1"></div>
<div class="item2"></div>
```

效果如图 4.16 所示。

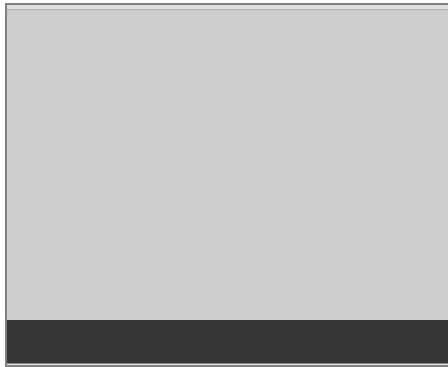


图 4.16 chrome 中的效果

但在火狐浏览器中是另外一个样子，如图 4.17 所示。



图 4.17 火狐浏览器中的效果

代码如下。

```
<style>  
  body,div{  
    margin:0;  
    padding:0;  
  }  
  .item1{  
    position:fixed;  
    left:0;  
    bottom:0;  
    width:100%;  
    height:60px;  
    background-color:red;  
  }  
  .item2{  
    width:100%;  
    height:1000px;  
    margin-bottom:50px;
```

```

        background-color: pink;
    }
</style>
<div class="item1"></div>
<div class="item2"></div>

```

其实只需要把 `margin-bottom` 改成 `fixed` 元素的高度即可, 这里把 `item2` 的 `margin-bottom` 改成 `60px` 就可以了, 如图 4.18 所示。



图 4.18 把 `margin-bottom` 改成 `fixed` 元素的高度

经过测试, 使用 IE 浏览器和火狐浏览器出现的情况是一样的。不过 `margin-bottom` 在 IE 7 浏览器以下的版本中不起作用。可以将代码进行修改, 代码如下。

```

<style>
    body,div{
        margin:0;
        padding:0;
    }
    .item1{

```



```
position:fixed;
left:0;
bottom:0;
width:100%;
height:60px;
background-color:red;
}
.item2{
width:100%;
height:1000px;
background-color:pink;
}
.line{
height:60px;
}
</style>
<div class="item1"></div>
<div class="item2"></div>
<div class="line"></div>
```

效果如图 4.19 所示。



图 4.19 给最后再添加一个 div

`position:fixed` 在 Chrome 浏览器中，实际上包括那个滚动条的高度。而在 IE 浏览器和火狐浏览器中是不包括的，在这两个浏览器中滚动条的高度是 15px。因此在 Chrome 浏览器中，只需要把 `fixed` 的高度减去 15px 就可以了。

`fixed` 会被固定住，而 `absolute` 不会。没有滚动之前，如图 4.20 所示。滚动之后，如图 4.21 所示。



图 4.20 没有滚动之前

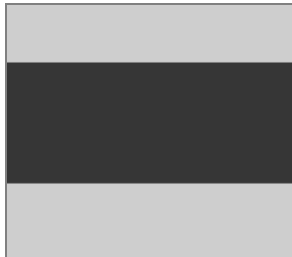


图 4.21 滚动之后

可以看到 `position` 被“滚”走了，而 `fixed` 的元素却还存在，代码如下。

```
<style>
body,div{
    margin:0;
    padding:0;
}
.item1{
    position:fixed;
    left:0;
    top:50px;
    width:100%;
    height:60px;
    background-color:red;
}
.item2{
    position:absolute;
    left:0px;
    top:100px;
    z-index:-1;
    width:100%;
    height:1000px;
    background-color:pink;
}
```

```
    }  
</style>  
<div class="item1"></div>  
<div class="item2"></div>
```

因为 item2 在 item1 后面，为了防止 item2 把 item1 盖住，所以加了“z-index:-1;”。

4.1.6 定位之 static 篇

将一个元素的 position 属性设置为 static，和不设置的效果是一样的。因为当没有设置 position 属性时，默认 position 属性的值就是 static。在一个原本你想定位，后来又不想要定位的元素上就可以使用这个，一般会配合 JavaScript 来用。当然，也可以让 position 的值为空，但使用 static 来取消定位更加合理，效果如图 4.22 所示。

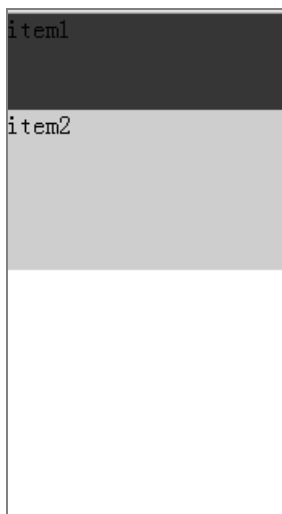


图 4.22 使用 static 取消定位

代码如下。

```
<style>  
  body,div{  
    margin:0;  
    padding:0;  
  }  
  .item1{  
    position:static;  
  }
```

```

    top:100px;
    left:100px;
    width:100%;
    height:60px;
    background-color:red;
  }
  .item2{
    width:100%;
    height:100px;
    background-color:pink;
  }
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>

```

4.2 透彻研究定位隐藏的秘密

在前面讲的内容中，大部分情况都是使用 top、left，接下来看看 bottom 和 right，如图 4.23 所示。



图 4.23 bottom 和 right

代码如下。

```

<style>
  body,div{

```

```
margin:0;
padding:0;
}
.item1{
position:absolute;
bottom:50px;
right:100px;
width:100%;
height:60px;
background-color:red;
}
</style>
<div class="item1">item1</div>
```

如果 `right` 值为正，那么就是距离页面右边的距离。如果仔细观察会发现，`item1` 里面的内容看不到了，主要是因为 `item1` 的宽度设置的是 `100%`，然后又距离右边 `100px`，所以总的宽度就是“`100% + 100px`”。因此，还有一部分被隐藏到左边了。

如果同时设置 `left`、`top`、`bottom`、`right` 会发生什么呢？如果同时设置 `left`、`top`、`bottom`、`right`，那么只有 `left` 和 `top` 值起作用，效果如图 4.24 所示。



图 4.24 同时设置 `left`、`top`、`bottom`、`right`

代码如下。

```
.item1{
position:absolute;
left:50px;
```

```
top:50px;  
right:50px;  
bottom:50px;  
width:100%;  
height:50px;  
background-color:red;  
}
```

如果同时设置 left、top、bottom、right，又想让 right 或者 bottom 起作用。那么可以将 left 设置为 auto，代码如下。

```
.item1{  
position:absolute;  
left:auto;  
top:50px;  
right:50px;  
bottom:50px;  
width:100%;  
height:50px;  
background-color:red;  
}
```

效果如图 4.25 所示。



图 4.25 将 left 设置 auto

当 left、top、right、bottom 都设置为 auto 时，如果它上面没有其他元素，相当于没有进行设置，效果如图 4.26 所示。



图 4.26 当 left、top、right、bottom 都设置为 auto 时的效果

代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .item1{
    position:absolute;
    left:auto;
    top:auto;
    right:auto;
    bottom:auto;
    width:100%;
    height:50px;
    background-color:red;
  }
</style>
<div class="item1">item1</div>
```

当父元素值为 static 时，效果如图 4.27 所示。



图 4.27 父元素值为 static

代码如下。

```
.box{
  position:static;
  margin-left:100px;
  width:200px;
  height:200px;
  border:1px solid #dedede;
}
```

因为当父元素值设置为 static 时，就相当于没有定位。

当同时设置 margin-left 和 left 时，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    position:absolute;
    left:100px;
    margin-left:100px;
    width:200px;
    height:200px;
    border:1px solid #dedede;
  }
</style>
<div class="box"></div>
```

效果如图 4.28 所示。



图 4.28 当同时设置 margin-left 和 left 时

显示的内容距离左边是 200px，也就是说，当设置 `margin-left` 和 `left` 时，距离左边的距离就是 `margin-left+left` 的结果。

结合 `margin:auto` 垂直居中，效果如图 4.29 所示。

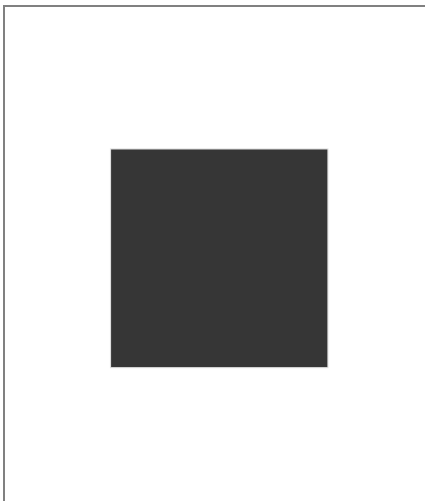


图 4.29 `margin:auto` 实现垂直居中

代码如下。

```
<style>
body,div{
    margin:0;
    padding:0;
}
.box{
    position:absolute;
    left:0;
    top:0;
    right:0;
    bottom:0;
    margin:auto;
    width:200px;
    height:200px;
    background-color:red;
    border:1px solid #dedede;
}
```

```
</style>  
<div class="box"></div>
```

通过 fixed 实现遮罩层，效果如图 4.30 所示。

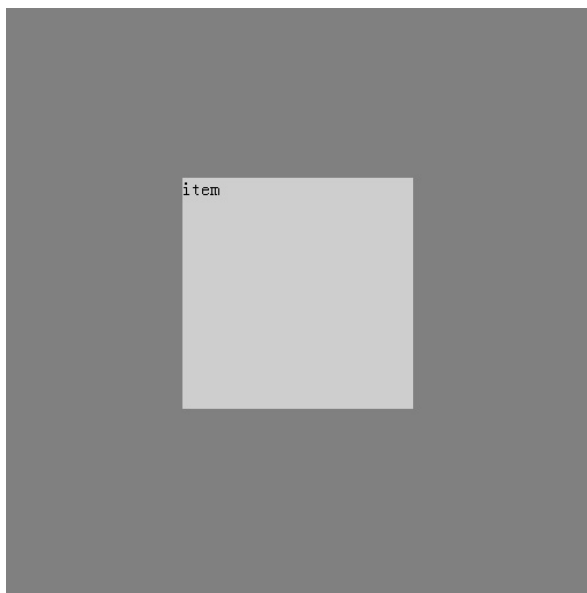


图 4.30 遮罩层

代码如下。

```
<style>  
  body,div{  
    margin:0;  
    padding:0;  
  }  
  .box{  
    position:fixed;  
    left:0;  
    top:0;  
    width:100%;  
    height:100%;  
    background-color:rgba(0,0,0,0.5);  
  }  
  .item{  
    position:absolute;
```

```
    left:0;
    top:0;
    right:0;
    bottom:0;
    width:200px;
    height:200px;
    margin:auto;
    background-color:pink;
}
</style>
<div class="box">
  <div class="item">item</div>
</div>
```

包不住的 fixed，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    position:absolute;
    left:100px;
    width:200px;
    height:200px;
    border:1px solid #dedede;
  }
  .item{
    position:fixed;
    left:0;
    width:100px;
    height:100px;
    background-color:red;
  }
</style>
<div class="box">
  <div class="item"></div>
</div>
```

效果如图 4.31 所示。导致父元素包不住定位值为 `fixed` 的元素的原因是 `fixed` 的包含块是视口，如果能让 `fixed` 根据它的父元素定位，那么就不能设置 `left`、`bottom`、`top`、`right` 值（除非值是 `auto`）。

代码如下。

```
.item{
  position:fixed;
  width:100px;
  height:100px;
  background-color:red;
}
```

将 `left`、`right`、`top`、`bottom` 值设置成 `auto` 和不设置是一样的，效果如图 4.32 所示。

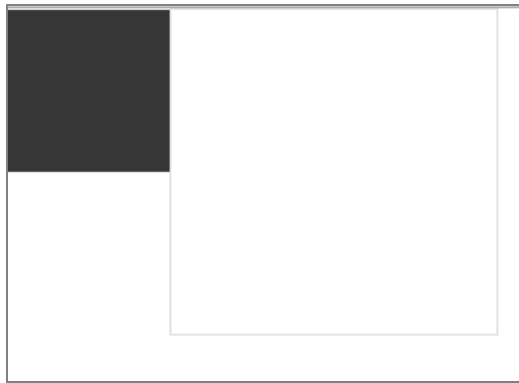


图 4.31 包不住的 `fixed`

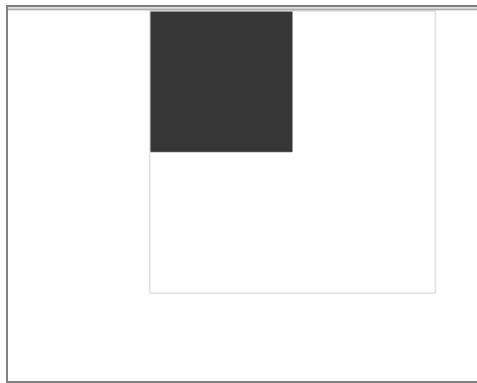


图 4.32 将 `left`、`right`、`top`、`bottom` 值设置成 `auto`

代码如下。

```
.item{
  position:fixed;
  left:auto;
  right:auto;
  top:auto;
  bottom:auto;
  width:100px;
  height:100px;
  background-color:red;
}
```

如果没有设置 `left`、`top`、`right`、`bottom`，那么默认值就是 `auto`。

一旦定位值为 `absolute`、`fixed`，高度将百分之百生效，代码如下。

```
.box{
  position:absolute;
  left:0;
  top:0;
  width:100%;
  height:100%;
  background-color:pink;
}
<div class="box"></div>
```

`fixed` 的百分比参照视口计算，不管是否有父元素。`absolute` 如果在一个定位的父元素里面，那么则按照父元素计算后的值进行计算，否则参照初始包含块。

父元素无法自适应定位元素的高度，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    position:relative;
    width:300px;
    border:10px solid #dedede;
  }
  .item{
    position:absolute;
    left:0;
    top:0;
    width:100px;
    height:100px;
    background-color:pink;
  }
</style>
<div class="box">
  <div class="item"></div>
</div>
```

效果如图 4.33 所示。



图 4.33 父元素无法自适应定位元素的高度

目前还没有解决方案，如果需要实现父元素自适应子元素的高度，请使用其他方式。

`fixed` 的另外一个问题就是当在移动端使用 `position:fixed` 时，制作遮罩时会发现，在部分浏览器中，如果滚动内容，遮罩后面的内容也会跟着滚动。要解决这个问题，可以使用一个小技巧，代码如下。

```
body{  
    position:fixed;  
    left:0;  
    top:0;  
}
```

在显示遮罩时给 `body` 也加一个固定定位，那么它就不会滚动了因为 `fixed` 的高度是参照一个视口的高度，但这同时也会引发另外一个问题，那就是需要滚动的内容也滚动不了，如果想解决这个问题，可以通过给固定定位的那个元素加上以下这段代码。

```
overflow-y:auto;
```

给需要滚动的元素加一个 `overflow-y:auto` 就可以了，其实更简单的做法是，当显示遮罩时给 `body` 加 “`overflow:hidden;`”，然后关闭遮罩时再去掉就可以了。

4.3 总结

- 除了 `position` 值为 `static` 和 `relative` 时不会脱离标准流以外，其他的几个值都会脱离标准流。虽然 `relative` 不会脱离标准流，但是拥有定位的一些其他特性（如果是行内元素，宽度、高度等还是不能用的）；
- `absolute` 和 `fixed` 的区别在于，定位值为 `fixed` 的元素始终都是根据视口来定位的。

`fixed` 的另外一个特点就是不会随着滚动而滚动，并且定位的属性（除 `static` 以外）如果没有设置 `left`、`top`、`bottom`、`right`，则总是在没有定位之前的位置上脱离标准流；

- 在一般情况下都会将 `absolute` 的元素放在 `relative` 里面，这样它总是跟随父元素走。

第 5 章 元素的七十二变——元素转换

5.1 display 介绍

display 常用的 4 种定位属性如下。

- none: 在格式化结构中不产生框，同时使子元素也不产生任何框；
- block: 生成一个块框，独占一行，在没有设置高度的情况下，高度由内容决定；
- inline: 生成一个或多个行内框，其他行内元素或行内块，以及浮动可以与它并排，宽高由内容决定，并且 width、height、margin-top、margin-bottom 将不起作用，可以使用 padding，但上下 padding 不占位置；
- inline-block: 其他行内元素或行内块，以及浮动可以和它并排，宽高由内容决定，但可以使用 width、height、margin、padding 等属性。

初始值为 inline，不过尽管初始值为 inline，但客户端的缺省样式表仍然可以超越它，可以发现一些浏览器不认识的元素会被当作行内元素就是这个原因。而 div 等被当作块元素，是因为客户端的默认样式覆盖了它。

5.2 大块头——block

就算 block 设置宽度，其后面的元素也不能和 block 并排，代码如下，效果如图 5.1 所示。

```
.item1{  
    display:block;
```



```
width:200px;
background-color:pink;
}
<div class="item1">item1</div>
<span class="item2">item2</span>
```



图 5.1 block 特性

5.3 我们一起站一排——inline

一旦将一个块级元素设置成 inline, 那么它将会和其他行内以及行内块并排, 代码如下。

```
<style>
body,div{
margin:0;
padding:0;
}
div{
display:inline;
background-color:red;
}
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>
```

效果如图 5.2 所示。

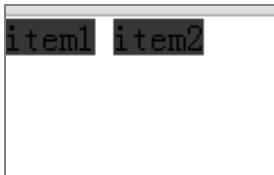


图 5.2 inline 可以和其他行内以及行内块并排

但是它会有几像素的间距，解决方法是把这两个元素放在一行，代码如下。

```
<div class="item1">item1</div><div class="item2">item2</div>
```

效果如图 5.3 所示。



图 5.3 将两个元素放在一行

这个间距与 `display` 无关，是由换行或者回车导致的，另外，将 `display` 的值设置为 `inline` 后，`margin-top`、`margin-bottom`、`height`、`width` 将不起作用，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .item1{
    display:inline;
    width:200px;
    height:200px;
    margin:50px;
    background-color:red;
  }
</style>
<div class="item1">item1</div>
<span>inline</span>
<div class="item2">item2</div>
```

效果如图 5.4 所示。

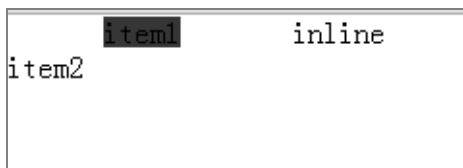


图 5.4 不起效果的属性

`inline` 元素对 `padding` 的解析，代码如下。

```

<style>
  body,div{
    margin:0;
    padding:0;
  }
  .item1{
    display:inline;
    padding:100px;
    background-color:pink;
  }
  .item2{
    width:100px;
    height:100px;
    background-color:green;
    border:1px solid #dedede;
  }
</style>
<div class="item1">inline padding</div>
<div class="item2">block</div>

```

效果如图 5.5 所示。

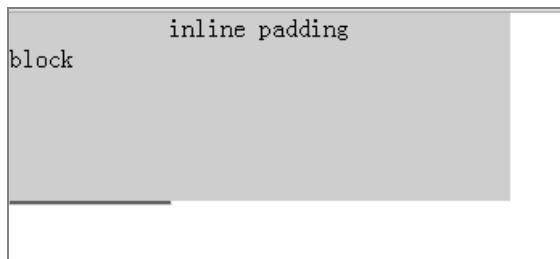


图 5.5 inline 怪异的 padding

inline 的 padding 会起作用，不过就像脱离了标准流一样，并不会占据位置，并且还把其他元素给盖住了。除了文字以外，看似 padding-top 没有起作用，但其实是因为它不占位置的原因，跑到浏览器窗口外面去了，如果将代码修改一下，就可以看到 padding-top 部分了，代码如下。

```

<style>
  body,div{
    margin:0;

```

```
padding:0;
}
.box{
    height:120px;
    background-color:red;
}
.item1{
    display:inline;
    padding:100px;
    background-color:pink;
}
.item2{
    width:100px;
    height:100px;
    background-color:green;
    border:1px solid #dedede;
}
</style>
<div class="box">box</div>
<div class="item1">inline padding</div>
<div class="item2">block</div>
```

效果如图 5.6 所示。

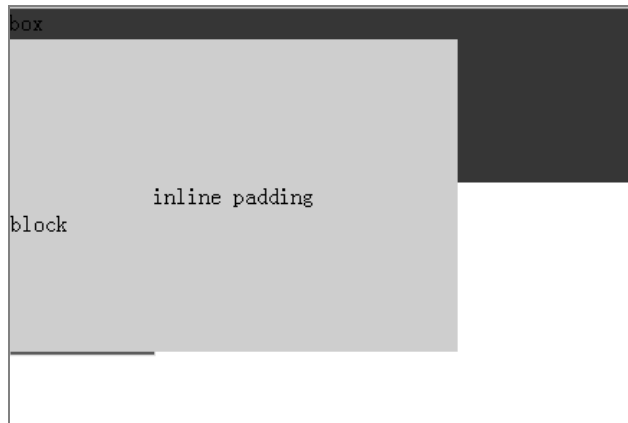


图 5.6 inline 的 padding-top、padding-bottom 不占位置

假如 inline 的元素没有内容，“padding-top、padding-bottom”将不起作用，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .item1{
    height:100px;
    background-color:red;
  }
  .item2{
    display:inline;
    padding-top:50px;
    padding-bottom:50px;
    background-color:pink;
  }
</style>
<div class="item1"></div>
<div class="item2"></div>
```

效果如图 5.7 所示。



图 5.7 假如 inline 的元素没有内容

如果想让 top 和 bottom 起作用，只需要给 left 或者 right 设置一个值，或者当 inline 的元素有内容时就会起作用，代码如下。

```
.item2{
  display:inline;
  padding-top:50px;
  padding-bottom:50px;
```

```
padding-left:50px;
background-color:pink;
}
```

效果如图 5.8 所示。



图 5.8 给 left 或者 right 设置一个值

当有内容时，代码如下，效果如图 5.9 所示。

```
.item2{
  display:inline;
  padding-top:50px;
  padding-bottom:50px;
  background-color:pink;
}
<div class="item1"></div>
<div class="item2">item2</div>
```



图 5.9 有内容时的效果

这里可以总结出 inline 的 padding 不占位置，并且 padding 的层级比其他元素的高。

5.4 inline 和 block 的结合体——inline-block

inline-block 拥有 inline 的并排特性，同时拥有 block 的宽高、margin 等特性，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    display:inline-block;
    width:50px;
    height:50px;
    padding:50px;
    margin:50px;
  }
  .item1{
    background-color:red;
  }
  .item2{
    background-color:pink;
  }
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>
```

效果如图 5.10 所示。

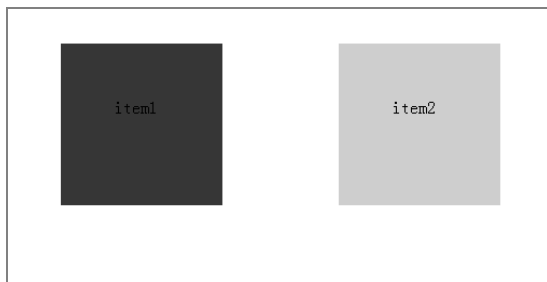


图 5.10 inline-block

inline-block 的高度和宽度由内容决定，代码如下。

```
.item1{
    display:inline-block;
    background-color:red;
}
<div class="item1">item1</div>
```

效果如图 5.11 所示。

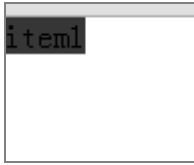


图 5.11 高度和宽度由内容决定

当 inline-block 碰到同类（inline、inline-block）时，代码如下。

```
<style>
    body,div{
        margin:0;
        padding:0;
    }
    .item1{
        display:inline-block;
        width:100px;
        height:100px;
        line-height:100px;
        background-color:red;
    }
    .item2{
        display:inline-block;
        background-color:pink;
    }
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>
```

效果如图 5.12 所示。

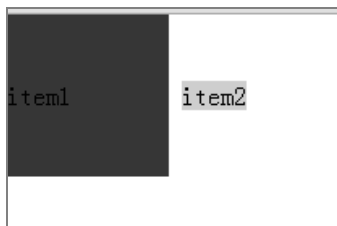


图 5.12 跟随

行高一起用，甚至 padding-top 和 margin-top 也一起用的效果如图 5.13 所示。



图 5.13 padding-top 和 margin-top 一起用

代码如下。

```
.item1{
  display:inline-block;
  width:100px;
  height:100px;
  margin:50px;
  background-color:red;
}
```

谁的上下 margin、padding 或 line-height 大，就听谁的。除非它是 inline，因为 inline 的 margin 是不起效果的。且 inline 的 padding 是不占空间的，如图 5.14 所示。

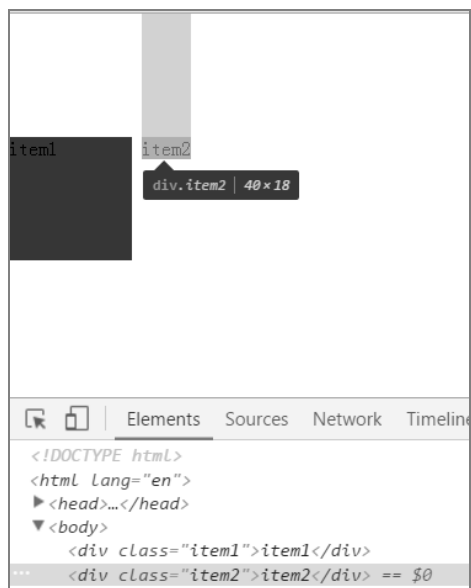


图 5.14 谁的上下 margin、padding 或 line-height 大，就听谁的

代码如下。

```

.item1{
  display:inline-block;
  width:100px;
  height:100px;
  margin-top:20px;
  background-color:red;
}
.item2{
  margin-top:100px;
  display:inline-block;
  background-color:pink;
}

```

5.5 行内和块的烦恼

我们发现 inline 和 inline-block 都会引起间距的空格，而我们又经常需要用到这个属性，下面来看看有哪些技术可以解决这个问题。

第1种方法是直接将空格删除；第2种方法是在空格代码中加入注释，代码如下。

```
.item1{
  display:inline-block;
  width:100px;
  height:100px;
  background-color:red;
}
.item2{
  display:inline;
  background-color:pink;
}
<div class="item1">item1</div><!--
  --><div class="item2">item2</div>
```

第2种方法是设置 `margin-left` 为负值，代码如下。

```
.item2{
  margin-left:-8px;
  display:inline;
  background-color:pink;
}
```

但这种方法有缺陷，就是每个浏览器的间距不一样，所以尽量不要用这个方法。

第3种方法是给父元素增加 “`font-size:0px;`”，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  body{
    font-size:0px;
  }
  .item1{
    display:inline-block;
    width:100px;
    height:100px;
    font-size:20px;
    background-color:red;
  }
```

```
.item2{
    display:inline-block;
    font-size:20px;
    background-color:pink;
}
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>
```

第 4 种方法是给父元素加 `letter-spacing` 负值，然后通过子元素清除 `letter-spacing` 值，代码如下。

```
<style>
body,div{
    margin:0;
    padding:0;
}
body{
    letter-spacing:-8px;
}
body *{
    letter-spacing:0px;
}
.item1{
    display:inline-block;
    width:100px;
    height:100px;
    background-color:red;
}
.item2{
    display:inline-block;
    background-color:pink;
}
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>
```

这里我们给 `body` 增加 `letter-spacing`，并且使用了一个通配符`*`，后面我们就不必为这个间距发愁了。

5.6 display 的一些其他属性

list-item 元素会作为列表显示，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box>.item{
    display:list-item;
    list-style-type:cjk-earthly-branch;
    list-style-position:inside;
  }
</style>
<div class="box">
  <div class="item">item</div>
  <div class="item">item</div>
  <div class="item">item</div>
</div>
```

将一个元素的 display 属性设置为 list-item，可以使用 list 的一些属性，效果如图 5.15 所示。

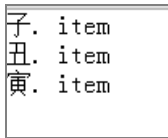


图 5.15 list-item

利用 “display:table;” 模拟 table 特性，以下是 “display:table” 的一些属性。

- table: 此元素会作为块级表格来显示；
- inline-table: 此元素会作为内联表格来显示；
- table-row-group: 此元素会作为一个或多个行的分组来显示；
- table-header-group: 此元素会作为一个或多个行的分组来显示；
- table-footer-group: 此元素会作为一个或多个行的分组来显示；

- `table-row`: 此元素会作为一个表格行显示;
- `table-column-group`: 此元素会作为一个或多个列的分组来显示;
- `table-column`: 此元素会作为一个单元格列显示;
- `table-cell`: 此元素会作为一个表格单元格显示;
- `table-caption`: 此元素会作为一个表格标题显示。

以下是利用 `display` 制作表格的例子, 代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .table{
    width:500px;
    display:table;
    border-collapse: collapse;
  }
  .table .row{
    display:table-row;
  }
  .table .cell{
    display:table-cell;
    height:35px;
    line-height:35px;
    border:1px solid #dedede;
    text-align:center;
  }
  .table .caption{
    display:table-caption;
    height:35px;
    line-height:35px;
    text-align:center;
  }
</style>
<div class="table">
  <div class="caption">这是一个表格</div>
  <div class="row">
```

```

<div class="cell">item1</div>
<div class="cell">item1</div>
<div class="cell">item1</div>
</div>
<div class="row">
  <div class="cell">item2</div>
  <div class="cell">item2</div>
  <div class="cell">item2</div>
</div>
</div>

```

效果如图 5.16 所示。

这是一个表格		
item1	item1	item1
item2	item2	item2

图 5.16 利用 display 制作表格

利用表格的特性，可以很容易地实现内容自适应，效果图 5.17 所示。

这是一个表格		
item1	item1	item1
item2	item2	item2

图 5.17 利用表格实现自适应

图 5.17 所对应的代码如下。

```

.table{
  width:100%;
  display:table;
  border-collapse: collapse;
  box-sizing:border-box;
}

```

因为加了 border，又用了 100%，防止总的宽度超出屏幕，所以这里使用“box-sizing: border-box;”让总的宽度包括 border，这样就不会超出屏幕了。

将显示内容制作成垂直水平居中的效果，效果图 5.18 所示。

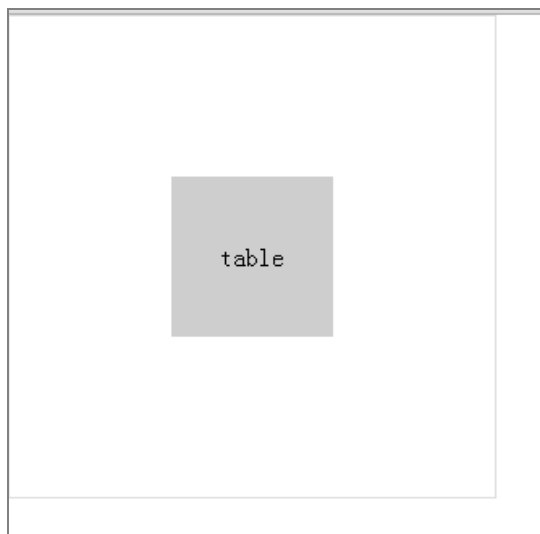


图 5.18 制作垂直水平居中

图 5.18 所对应的代码如下。

```
<style>
body,div{
    margin:0;
    padding:0;
}
.box{
    width:300px;
    height:300px;
    display:table;
    border:1px solid #dedede;
}
.box div{
    display:table-cell;
    vertical-align:middle;
}
.box div p{
    width:100px;
    height:100px;
    line-height:100px;
    margin:0 auto;
    text-align:center;
}
```



```
        background-color: pink;
    }
</style>
<div class="box">
    <div>
        <p>table</p>
    </div>
</div>
```

原理是使用 `display:table-cell` 的元素可以像表格一样使用 `vertical-align` 属性，先让它垂直居中，然后再给子元素设置 “`margin:0 auto;`” 就可以了。

`display` 还有一个 `flex` 的属性，用来实现弹性盒子，相关知识将在布局篇里讲解。

5.7 总结

- 在一个元素中使用 “`display:inline;`” 那么这个元素将拥有行内元素的特性，并且 `padding-top`、`padding-bottom` 不会占位置；
- `inline` 和 `inline-block` 元素之间都会有几像素的间距；
- 利用 `display:table` 可以很轻松地实现垂直水平居中；
- 利用 `display:list-item` 可以使用 `li` 的一些样式。

第 6 章 浮动趣事

浮动让我们对它又爱又恨，爱的是用它来实现布局实在是太方便了，恨的是用它来实现布局后带来的“麻烦”也很多，但它真的是用来布局的吗？

6.1 浮动简介

如果将一个元素设置为浮动（除 `none`），那么它将可以在当前行向左或向右偏移，并且内容可以在浮动元素的周围排列。

除 `none` 值以外，浮动的元素会产生一个块框，它会不断地向左或向右偏移，直到它的外边接触到包含块的边界或另外一个浮动的外边界。浮动框的顶部和当前线框的顶部对齐，如果没有线框，则对齐前面一个块框的底部。如果当前行没有足够的空间容纳它，那么它会逐行向下移动，直到某一行有足够的空间容纳它。

由于浮动框并不在标准流里，所以如果它后面的是块元素，那么块元素会当浮动框不存在一样。

浮动的框并不会出现在包含块的外边界上，并且它的顶边不可以高于它的包含块的顶，也不可以高于源文档中先前元素产生的块或浮动框的顶。

`float` 有以下 3 个属性。

- `none`: 不浮动，默认值；
- `left`: 产生一个块框，并向左浮动。起点是框的顶部（除非还设置了 `clear` 属性），`display` 被忽略，除非它的值是 `none`，如果它的后一个元素是块级元素，那么后一个元素将会占据浮动的那个元素的位置，并且浮动的元素层级总是在标准流元素上面；
- `right`: 和 `left` 一样，不过它是右浮动。

6.2 浮动的特点

下面来看看盖不住的文本，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    width:100px;
    height:100px;
  }
  .item1{
    float:left;
    background-color:red;
  }
  .item2{
    background-color:pink;
  }
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>
```

效果如图 6.1 所示。

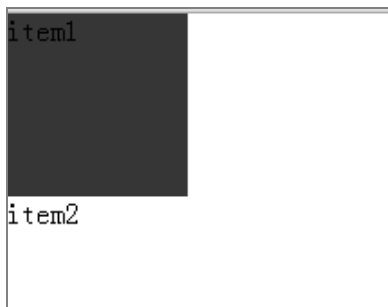


图 6.1 盖不住的文本

除了文本，其他的内容都看不见了，因为它跑到 `item1` 下面去了。

如果浮动的后面不是块级元素，后面的元素将会和它并排（除非设置了元素的宽度，并且屏幕放不下时将会换行），代码如下。

```
.item1{
  float:left;
  background-color:red;
}
.item2{
  display:inline-block;
  background-color:pink;
}
<div class="item1">item1</div>
<div class="item2">item2</div>
```

效果如图 6.2 所示。



图 6.2 浮动的后面不是块级元素

一般会使用盖不住文本的特性来实现左侧宽度固定，右侧自适应，代码如下。

```
<style>
body,div,p,h1{
  margin:0;
  padding:0;
}
.box .item1{
  width:100px;
  height:200px;
}
.item1{
  float:left;
  background-color:red;
}
```

```

.item2{
    height:200px;
    background-color:pink;
}
</style>
<div class="box">
    <div class="item1">item1</div>
    <div class="item2">
        <h1>自行车前灯骑行装备固定前灯支架</h1>
        <p>EDISON 自行车灯车前灯，三色可选，抗颠簸防水，可用 USB 充电，安装到车架上简单方便；
也可单独做手电使用，外形时尚美观，光照范围符合德国 STVZO 认证；特殊光学设计，像汽车灯一样，
可将凹凸不平的路面照得清清楚楚，是你外出夜骑的好选择！ </p>
    </div>
</div>

```

效果如图 6.3 所示。

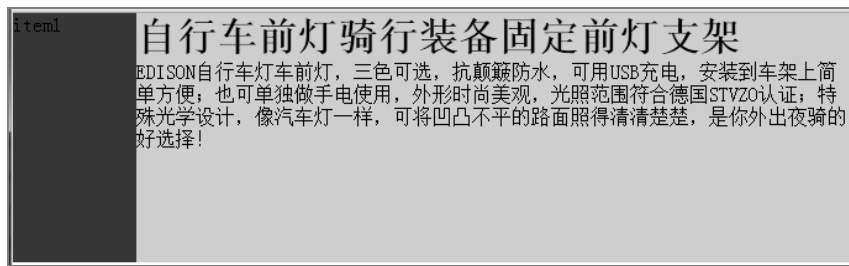


图 6.3 左侧宽度固定，右侧自适应

浮动只在当前行浮动（如果它的上一个元素没有浮动），代码如下。

```

<style>
    body,div{
        margin:0;
        padding:0;
    }
    div{
        width:100px;
        height:100px;
    }
    .item1{
        background-color:red;
    }

```

```

.item2{
    float:left;
    background-color:pink;
}
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>
<div class="item3">item3</div>

```

效果如图 6.4 所示。

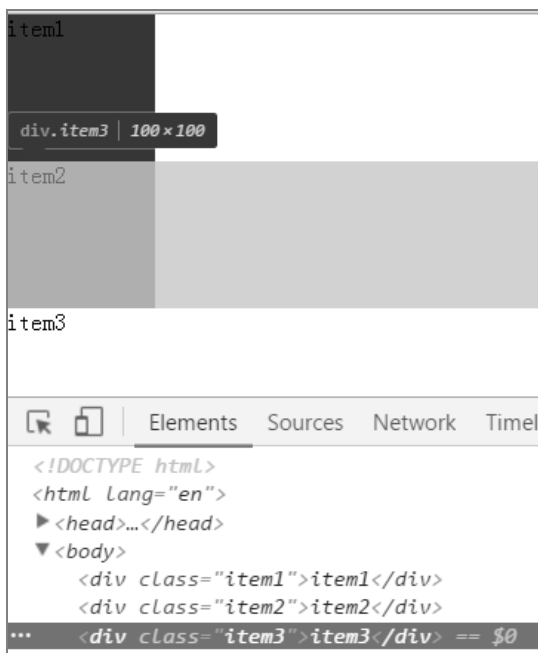


图 6.4 浮动只在当前行浮动

当浮动偶遇浮动，它们将在一行排序，除非没有位置了，代码如下。

```

<style>
body,div{
    margin:0;
    padding:0;
}
div{
    width:400px;

```

```
height:100px;
float:left;
}
.item1{
background-color:red;
}
.item2{
background-color:pink;
}
.item3{
background-color:green;
}
.item4{
background-color:orange;
}
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>
<div class="item3">item3</div>
<div class="item4">item4</div>
```

效果如图 6.5 所示。

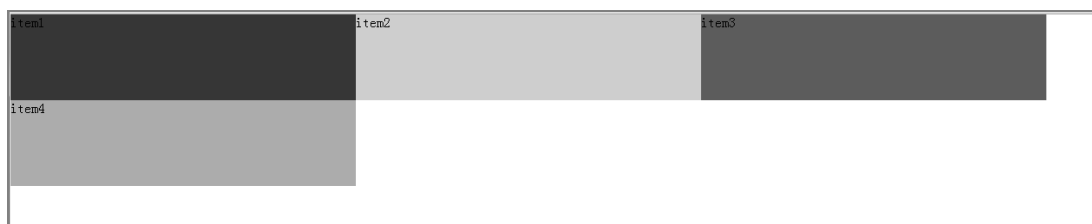


图 6.5 当浮动偶遇浮动

如果想实现自适应，可以用“100/份数”，代码如下。

```
div{
width:25%;
height:100px;
float:left;
}
```

效果如图 6.6 所示。

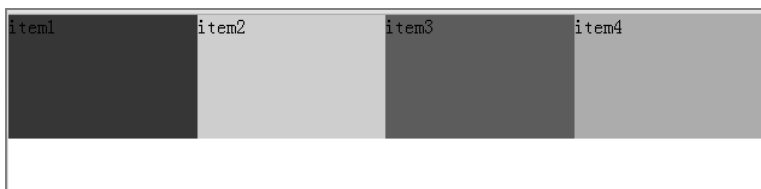


图 6.6 实现自适应

右浮动和左浮动的效果一样，不过是一个在左边，一个在右边，如图 6.7 所示，代码如下。



图 6.7 右浮动

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  div{
    width:100px;
    height:100px;
    line-height:100px;
    text-align:center;
  }
  .item1{
    float:right;
    background-color:pink;
  }
  .item2{
    background-color:red;
  }
</style>
<div class="item1">item1</div>
<div class="item2">item2</div>
```


当元素设置定位值为 `absolute`、`fixed` 时，浮动将被忽略，代码如下。

```
<style>
  div{
    position:absolute;
    float:right;
    width:100px;
    height:100px;
    border:1px solid red;
  }
</style>
<div></div>
```

效果如图 6.8 所示。

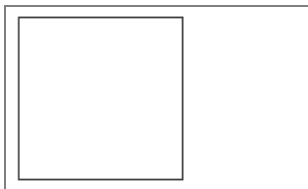


图 6.8 定位值为 `absolute`、`fixed` 时浮动将被忽略

6.3 浮动的秘密

因为行内元素使用浮动以后生成了一个块框，因此它可以使用 `width`、`height`、`margin`、`padding` 等属性，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  [class^="item"]{
    width:100px;
    height:100px;
    line-height:100px;
    text-align:center;
  }
```

```
.item1{
  float:left;
  background-color:pink;
}
.item2{
  display:inline-block;
  background-color:red;
}
</style>
<span class="item1">item1</span>
<div class="item2">item2</div>
```

效果如图 6.9 所示。

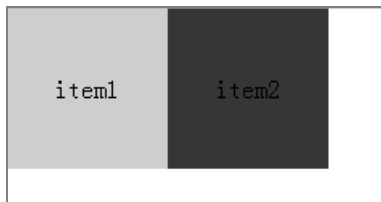


图 6.9 浮动的元素会生成一个块框

父元素无法自适应浮动元素的高度，代码如下。

```
<style>
body,div{
  margin:0;
  padding:0;
}
.item{
  float:left;
  width:100px;
  height:100px;
  background-color:pink;
}
</style>
<div class="box">
  <div class="item"></div>
</div>
```

效果如图 6.10 所示。

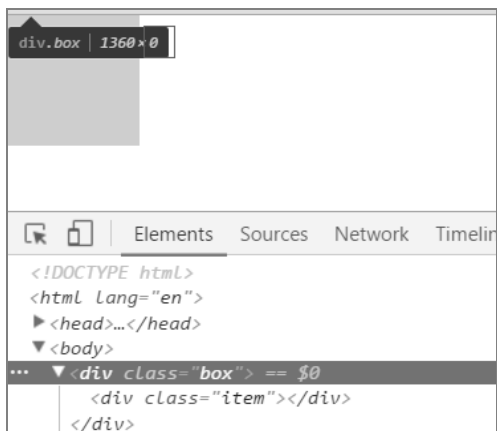


图 6.10 父元素无法自适应浮动元素的高度

如果想解决这个问题，可以使用以下几种方案。

第 1 种方案：给父元素增加 “overflow:hidden;”（只要 overflow 值不为 visible 就行），代码如下。

```
.box{  
    overflow:hidden;  
}
```

效果如图 6.11 所示。

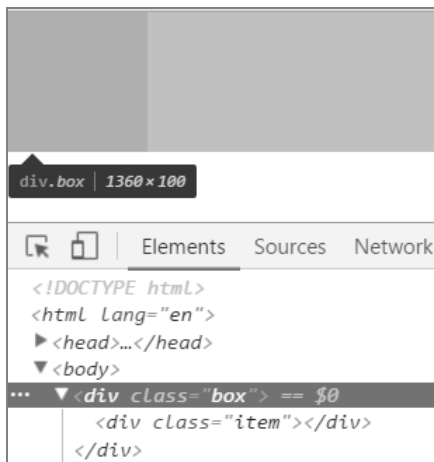


图 6.11 利用 overflow 清除浮动

第 2 种方案：在父元素的前后清除浮动，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box::after,.box::before{
    content:'';
    display:block;
    clear:both;
  }
  .item{
    float:left;
    width:100px;
    height:100px;
    background-color:pink;
  }
</style>
<div class="box">
  <div class="item"></div>
</div>
```

效果如图 6.12 所示。

浮动元素会被后一个元素的 margin-top 影响，如图 6.13 所示。



图 6.12 在父元素的前后清除浮动

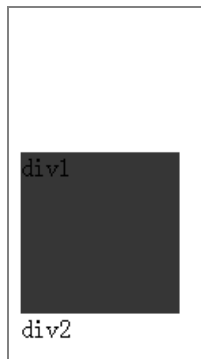


图 6.13 浮动元素会被后一个元素的 margin-top 影响

代码如下。

```
<style>
  div{
    width:100px;
    height:100px;
  }
  div:nth-of-type(1){
    float:left;
    background-color:red;
  }
  div:nth-of-type(2){
    margin-top:100px;
    background-color:pink;
  }
</style>
<div>div1</div>
<div>div2</div>
```

第一个 div 也跟着下来了。

若前一个元素浮动，则给后一个元素设置 clear 时，后一个元素的 margin-top 将无效，如图 6.14 所示。

代码如下。

```
<style>
  div{
    width:100px;
    height:100px;
  }
  div:nth-of-type(1){
    float:left;
    background-color:red;
  }
  div:nth-of-type(2){
    clear:both;
    margin-top:100000px;
    background-color:pink;
  }
</style>
```

```
<div>div1</div>
<div>div2</div>
```

想解决这个问题，给浮动的元素设置 `margin` 就可以了。
浮动的元素总是想更靠近父元素的顶部，如图 6.15 所示。

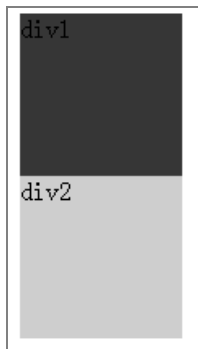


图 6.14 给元素设置 `clear` 时 `margin-top` 将无效

图 6.15 浮动的元素总是想更靠近父元素的顶部

这种问题出现的结果就是父元素和子元素都设置了 `line-height`，代码如下。

```
<style>
  div,p{
    margin:0;
  }
  .box{
    width:200px;
    height:200px;
    line-height:200px;
    border:1px solid #dedede;
  }
  .box span{
    float:right;
    height: 30px;
    line-height: 30px;
  }
</style>
<div class="box">
  <em>浮动来了</em>
  <span>逃</span>
</div>
```

效果如图 6.16 所示。



图 6.16 父元素和子元素都设置了 line-height

解决的办法就是不要给父元素加 line-height，而是加 padding，或者给浮动元素加 margin-top，或者干脆不用浮动。

6.4 实现任意形状的文字环绕

为什么文字不会被浮动的元素盖住呢？因为浮动的本质就是用来实现文字环绕的，如图 6.17 所示。

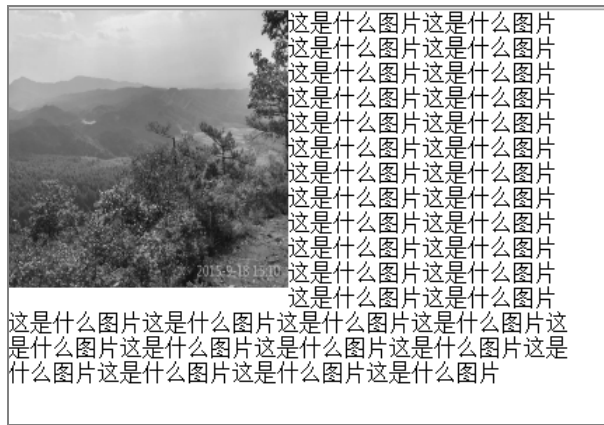


图 6.17 文字环绕

你会发现除了用浮动，没有其他更好的方式来实现这个效果了，代码如下。

```
<style>
body,div{
    margin:0;
    padding:0;
}
```

```
.box{
  width:400px;
}
img{
  width:200px;
  height:200px;
  float:left;
}
</style>
<div class="box">
  
</div>....</div>
```

说到文字环绕，不得不提一个新属性——**shape**，它可以用来设置文字环绕时的形状。**shape** 有以下两个属性。

- **shape-outside**：让文字从图片外部开始环绕，以及设置图片的形状；
- **shape-inside**：设置图片内部的形状，目前基本不支持。

可以用 **margin-box**、**content-box**、**border-box**、**padding-box** 关键字来设置从浮动的元素哪里开始环绕，还可以通过 **circle**、**ellipse**、**inset**、**polygon** 来定义基本形状。当然，它们也可以组合起来。

关键字（有点类似 **box-sizing**）如下。

- **margin-box**；
- **content-box**；
- **border-box**；
- **padding-box**。

基本形状用以下函数来定义。

- **circle()**：用于制作圆形形状；
- **ellipse()**：用于制作椭圆形状；
- **inset()**：用于制作矩形形状；
- **polygon()**：用于创建具有超过 3 个顶点的任何形状；
- **url()**：从图像中提取形状，只能提取透明的图片。

可以通过 `shape-outside` 设置，让文字在图片的内容中环绕，这样就不用在意图片设置的 `margin` 了，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:400px;
  }
  img{
    width:200px;
    height:200px;
    margin:10px;
    float:left;
    shape-outside:content-box;
  }
</style>
<div class="box">
  
  <div>....</div>
</div>
```

效果如图 6.19 所示。

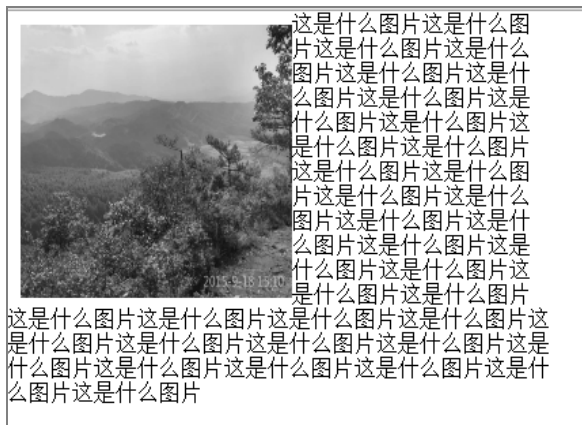


图 6.19 `shape-outside` 用来设置文字在图片中的环绕

实际上，只要值不是“margin-box;”都可以，就是看要让它从图片的哪里开始环绕。因为这是一张正方形的图片，所以看上去设置文字从图片哪里环绕都没有什么价值。但如果把图片变成一张圆形的图片呢？效果如图 6.20 所示。



图 6.20 当图片为圆形时

代码如下。

```
img{
  width:200px;
  height:200px;
  margin:10px;
  float:left;
  border-radius:50%;
  shape-outside:border-box;
}
```

以上代码完美地实现了文字与图片的环绕，也可以将图片设置成其他的形状。

还可以使用 `polygon` 多边形函数绘制一个图形，然后这些文字就会根据这个形状来进行环绕。

`polygon` 的语法如下。

```
polygon(x1 y1,x2 y2 x3 y3,...)
```

举个例子，代码如下。

```
<style>
  body,div{
    margin:0;
```

```
padding:0;
}
.box{
width:400px;
}
img{
width:200px;
height:200px;
float:left;
shape-outside:polygon(0 100%,50% 0,100% 100%);
}
</style>
<div class="box">
  
  <div>...</div>
</div>
```

效果如图 6.21 所示。

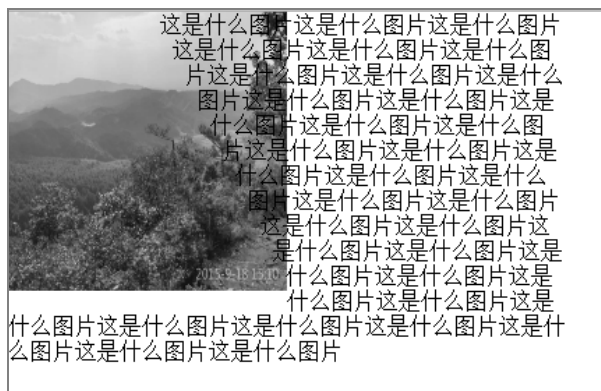


图 6.21 polygon 例子

`polygon` 的起点是根据图片的左上角来计算的，而百分比是根据元素的大小来计算的，比如 `x1` 的百分比为 100%，那么 `x1` 的值就是元素的宽度，50%就是宽度的一半，`y` 的百分比就是根据元素的高度。

实际上，上面的代码绘制的是一个三角形，如图 6.22 所示。

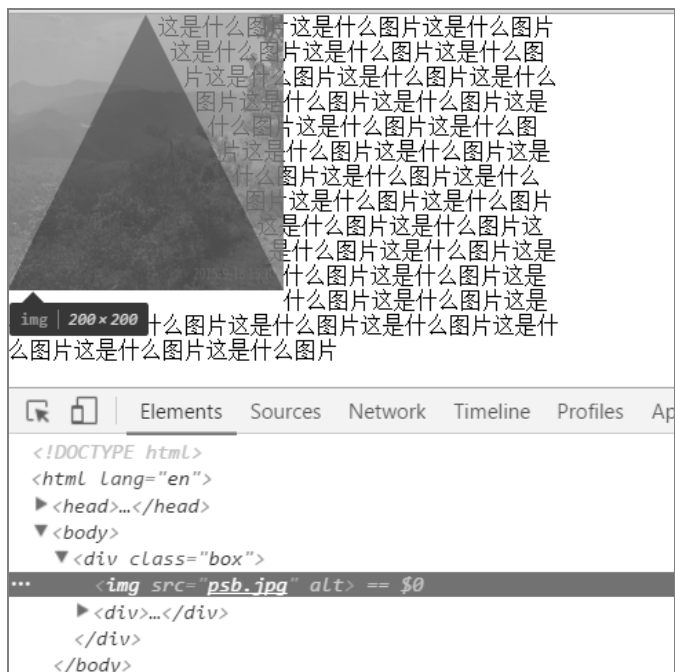


图 6.22 利用 polygon 绘制三角形

它里面的形状是一个三角形，`shape` 并不会改变图片的形状，只是改变文字环绕时的形状。如果想把图片的形状也改变，那么可以使用 `clip-path` 函数，代码如下。

```
img{
  width:200px;
  height:200px;
  float:left;
  shape-outside:polygon(0 100%,50% 0,100% 100%);
  -webkit-clip-path:polygon(0 100%,50% 0,100% 100%);
}
```

效果如图 6.23 所示。

`clip-path` 函数具有裁剪功能，因为浏览器兼容问题，所以这里加了“-webkit-”，这个函数在后面的章节里会详细讲解。

有时需要结合 `content-box` 等来使用，如图 6.24 所示。

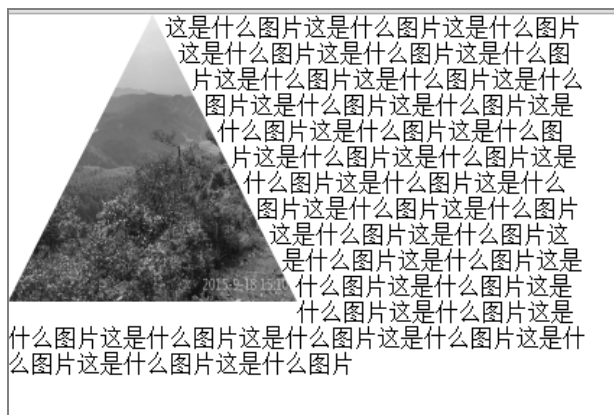


图 6.23 通过 clip-path 函数改变图片形状



图 6.24 结合 content-box 使用效果

此时只要在 shape-outside 后面加上 content-box 等就可以了，代码如下。

```
img{
  width:200px;
  height:200px;
  margin-right:20px;
  float:left;
  shape-outside:polygon(0 100%,50% 0,100% 100%)
  content-box;
  -webkit-clip-path:polygon(0 100%,50% 0,100% 100%);
}
```

将图片形状改为平行四边形，代码如下。

```
img{
  width:200px;
  height:200px;
  float:left;
  shape-outside:polygon(0 100%,50% 0,100% 0,50% 100%
,0 100%);
  -webkit-clip-path:polygon(0 100%,50% 0,100% 0,50%
100%,0 100%);
}
```

效果如图 6.25 所示。



图 6.25 平行四边形

用 circle 函数实现一个圆形，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:400px;
  }
  img{
    width:200px;
    height:200px;
    float:left;
    shape-outside:circle(50%);
```

```

    -webkit-clip-path:circle(50%);
}
</style>
<div class="box">
    
    <div>...</div>
</div>

```

效果如图 6.26 所示。



图 6.26 用 circle 函数实现一个圆形

如果想更精确地绘制一个形状的图形，可以使用 at 关键词，代码如下。

```

img{
    width:200px;
    height:200px;
    float:left;
    shape-outside:circle(50% at 0 0);
    -webkit-clip-path:circle(50% at 0 0);
}

```

效果如图 6.27 所示。

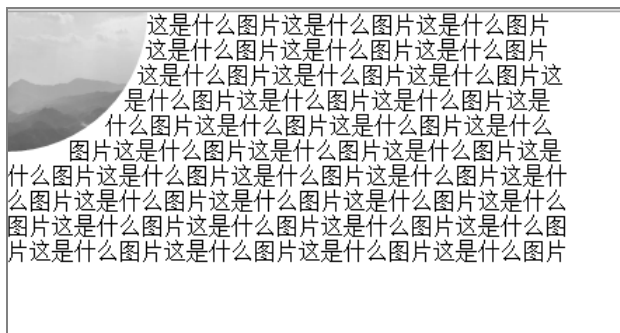


图 6.27 使用 at 关键词

第一个传的是 x，第二个传的是 y，可以只传一个。

用 ellipse 函数绘制一个椭圆形，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:400px;
  }
  img{
    width:200px;
    height:200px;
    float:left;
    shape-outside:ellipse(80%);
  }
</style>
<div class="box">
  
  <div>...</div>
</div>
```

但是你会发现，好像文字形状没什么变化，如图 6.28 所示。



图 6.28 用 ellipse 函数绘制的效果

可以确定的是形状没有问题，那么肯定就是因为这个形状超出了图片的大小了。也就是说，如果绘制的形状超出了图片，那么超出的那部分形状是不会影响到文字环绕的效果的。

减小百分比看看，代码如下，效果如图 6.29 所示。

```
img{
  width:200px;
  height:200px;
  float:left;
  shape-outside:ellipse(60%);
}
```



图 6.29 超出的那部分形状不会影响到文字环绕

使用 `inset` 函数绘制一个矩形形状，它接受 5 个参数，即 `top`、`right`、`bottom`、`left`、`round`，第 5 个参数是边界半径（可选）。其他的参数都是从边缘向内偏移。前面几个参数可以缩写，其格式与 `padding`、`margin` 一模一样。

代码如下。

```

<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:400px;
  }
  img{
    width:200px;
    height:200px;
    float:left;

```

```

    shape-outside:inset(22px);
  }
</style>
<div class="box">
  
  <div>...</div>
</div>

```

效果如图 6.30 所示。



图 6.30 用 inset 函数绘制一个矩形形状

加上一个边框，代码如下。

```

img{
  width:200px;
  height:200px;
  float:left;
  shape-outside:inset(22px round 20%);
}

```

效果如图 6.31 所示。



图 6.31 加上一个边框

从图片中提取形状，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:400px;
  }
  img{
    width:200px;
    height:200px;
    float:left;
    shape-outside: url(1.png);
    shape-image-threshold: 0.5;
  }
</style>
<div class="box">
  
```

```
<div>...</div>  
</div>
```

效果如图 6.32 所示。



图 6.32 从图片中提取形状

`shape-image-threshold` 用于创建形状像素的最小不透明度级别，值只能是 0 到 1 之间。

如果要使用 `shape`，那么元素必须是浮动的，且元素必须有宽和高。

6.5 总结

- 当浮动后面的元素是 `inline` 或 `inline-block` 时，它们会并排在一起，并且浮动的元素不会把它们盖住；
- 如果是 `block` 元素，则会把它给盖住，如果位置不够了会换行；
- 父元素无法自适应浮动元素的高度；
- 浮动的本质是用来实现文字环绕的，所以不会把文字盖住。

第 7 章 再不学这些选择器就老了

合理地使用选择器，可以帮你减轻很多工作。虽然选择器有很多，但大部分人还是停留在 ID、class 上，另外 CSS 3 新增了一些强大的选择器，你会发现利用它们可以替代 JavaScript 的一部分工作。

7.1 那些被遗忘的选择器

除了 ID、class 以外，还有很多强大的选择器。

7.1.1 相邻兄弟选择器

选择紧接在另一个元素之后的第一个元素（同级元素），代码如下。

```
<style>
  h1 + p{
    color:red;
  }
</style>
<h1>兄弟快过来</h1>
<p>来了</p>
<p>来了</p>
```

效果如图 7.1 所示。

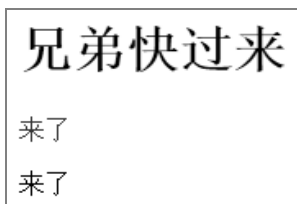


图 7.1 相邻兄弟选择器

可以看到，只选择到了 `h1` 之后的第一个兄弟元素。接着看下面的这个问题，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:400px;
  }
  .box > div{
    width:100px;
    height:35px;
    line-height:35px;
    text-align:center;
    float:left;
    border:1px solid #dedede;
    box-sizing:border-box;
  }
</style>
<div class="box">
  <div>追梦子</div>
  <div>CSS3</div>
  <div>HTML5</div>
  <div>Javascript</div>
</div>
```

效果如图 7.2 所示。

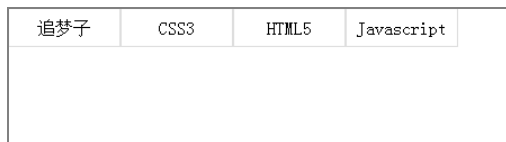


图 7.2 后面几个元素的左边框大了 1 像素

可以看到后面几个元素的左边框明显大了 1 像素。尽管用了 **box-sizing**，但它不是用来解决这个问题的，这里只需要用“+号”选择器把除第一个元素以外的左边框去掉就可以了，代码如下。

```
.box > div + div{
    border-left:none;
}
```

效果如图 7.3 所示。



图 7.3 利用“+号”选择器去除多出来的边框

如果再结合:checked 选择器，可以实现更多的效果，如图 7.4 所示。

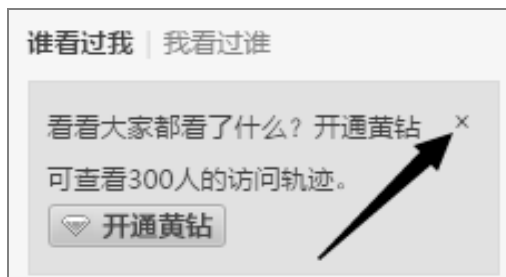


图 7.4 :checked 选择器

单击关闭对话框后，父级元素会被隐藏，代码如下。

```
<style>
body,div{
    margin:0;
    padding:0;
```

```

    }
    #close,#close:checked + label + .content,#close:checked + label{
        display:none;
    }
</style>
<div class="box">
    <input type="radio" id="close">
    <label for="close">关闭</label>
    <div class="content">
        这里充当父元素
    </div>
</div>

```

没单击之前的样子如图 7.5 所示。

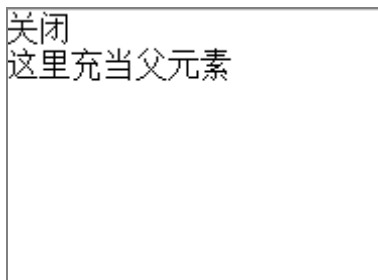


图 7.5 未单击之前

单击“关闭”的时候，整个元素就会被隐藏。

7.1.2 利用 hover 实现一个下拉菜单

利用 hover 实现下拉菜单，代码如下。

```

<style>
    body,div,ul,li{
        margin:0;
        padding:0;
    }
    ul{
        list-style:none;
    }
    .box{

```

```

        width:450px;
        margin:100px;
    }
    .box > ul li{
        width:150px;
        height:35px;
        line-height:35px;
        text-align:center;
        float:left;
        border:1px solid #dedede;
        box-sizing:border-box;
    }
    .box > ul > li li{
        border-top:none;
    }
    .box > ul > li + li{
        border-left:none;
    }
    .box > ul > li > ul{
        display:none;
    }
    .box > ul > li:hover ul{
        display:block;
    }
</style>
<div class="box">
    <ul>
        <li>
            追梦子
            <ul>
                <li>博客动态</li>
            </ul>
        </li>
        <li>
            CSS3
            <ul>
                <li>CSS3 文章</li>
                <li>CSS3 技巧</li>
            </ul>
        </li>
    </ul>
</div>

```

```

        </ul>
    </li>
    <li>
        HTML5
        <ul>
            <li>HTML5 最新动态</li>
            <li>HTML5 使用教程</li>
        </ul>
    </li>
</ul>
</div>

```

效果如图 7.6 所示。



图 7.6 利用 hover 实现下拉菜单

其中核心的代码如下。

```

.box > ul > li:hover ul{
    display:block;
}

```

此段代码的意思是，当用户移动到 li 上时，让 li 后面的 div 显示。

7.1.3 利用 active 做一个集能量

利用 active 做一个集能量，代码如下。

```

<style>
    body,div{
        margin:0;
        padding:0;
    }
    .box{

```

```

width:120px;
height:100px;
line-height:100px;
text-align:center;
margin:100px auto;
background-color:pink;
transition:all 1s;
}
.box:active{
padding:100px;
}
</style>
<div class="box">点击获取能量...</div>

```

效果如图 7.7 所示。

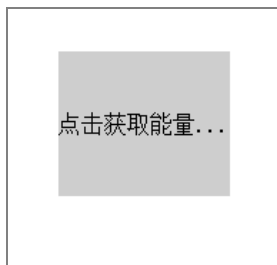


图 7.7 未单击时

使用一个过渡效果，代码如下。

```
transition:all 1s;
```

此行代码的意思是，当某些值改变时，它会慢慢地加到目标值，也就出现了过渡的效果。如果想详细了解过渡属性，可以参考此网站：https://developer.mozilla.org/zh-CN/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions。

7.1.4 用 first-letter 选中第一个字

利用 first-letter 选中第一个字，代码如下。

```

<style>
body,div{
margin:0;

```

```
padding:0;
}
.box{
    text-align:center;
    background-color:pink;
}
.box:first-letter{
    font-size:33px;
}
</style>
<div class="box">点击获取能量...</div>
```

效果如图 7.8 所示。

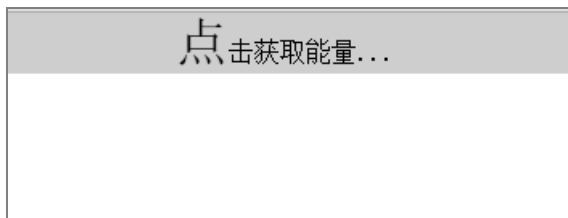


图 7.8 利用 first-letter 选中第一个字

还有一种特殊情况，代码如下。

```
<style>
body,div{
    margin:0;
    padding:0;
}
.box{
    margin-top:10px;
    margin-left:50px;
    background-color:pink;
}
.box:first-letter{
    font-size: 16px;
    line-height: 36px;
    margin:0px 6px;
    color: #fe7e01;
    background-color: currentColor;
```

```

    }
</style>
<div class="box">| 捷安特</div>
<div class="box">| 捷安特 ATX777</div>
<div class="box">| 捷安特 ATX830</div>

```

效果如图 7.9 所示。



图 7.9 左边竖线

背景用了一个 `currentColor` 关键字，它的颜色等于 `color` 的颜色。这样就不用再去复制粘贴了。如果碰到其他（背景、边框等）颜色和文字颜色一样时，就可以使用 `currentColor` 这个关键字。

不过这种方法也有一个缺点，就是不能精确地控制竖线的大小。因为 `first-letter` 这个选择器不允许使用宽度、高度等属性，而 `font-size` 又是按照比例来的。

7.1.5 用 `first-line` 选择首行文字

利用 `first-line` 选择首行文字，代码如下。

```

<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    margin-top:10px;
    margin-left:50px;
  }
  .box:first-line{
    color:red;
  }

```

```
    }  
</style>  
<div class="box">  
    清晨我看到了一缕阳光...  
    <br>  
    然后起床了...  
</div>
```

效果如图 7.10 所示。

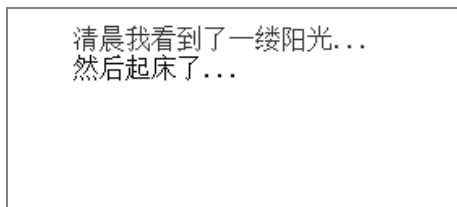


图 7.10 用 first-line 选择首行文字

还可以将代码修改如下。

```
<style>  
    body,div{  
        margin:0;  
        padding:0;  
    }  
    .box{  
        margin-top:10px;  
        margin-left:50px;  
    }  
    .box:first-line{  
        color:red;  
        font-size:20px;  
        line-height:2;  
    }  
</style>  
<div class="box">  
    春夏秋冬  
    <br>  
    春风送暖  
    <br>
```



```

    夏日炎炎
    <br>
    秋风习习
    <br>
    该冬眠了
</div>

```

效果如图 7.11 所示。

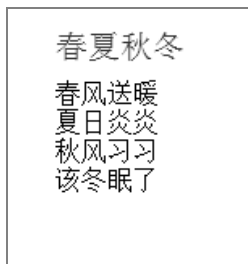


图 7.11 改变排列

7.2 模拟父级选择器

浏览器没有提供一个父级选择器，而有这种需求的人还是很多的。

当鼠标移动到 CSS 上的时候改变边框的颜色，以前多数时候都是用 JavaScript 来实现的，但其实 CSS 也可以实现，代码如下。

```

<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    position:relative;
    width:200px;
    height:200px;
    text-align:center;
    margin:100px auto;
  }
  .box .item2{

```

```

    position:absolute;
    left:0;
    top:0;
    width:100%;
    height:100%;
    outline:1px solid #dedede;
    z-index:-1;
}
.box .item1:hover + .item2{
    outline:1px solid red;
}
</style>
<div class="box">
  <div class="item1">
    CSS
  </div>
  <div class="item2"></div>
</div>

```

效果如图 7.12 所示。

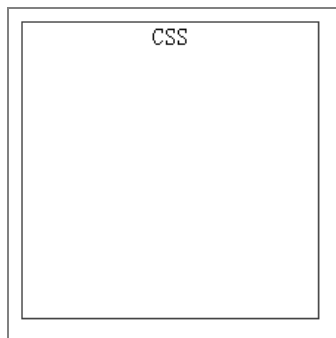


图 7.12 模拟父级选择器

这种效果的原理就是用其他元素替代父元素，然后通过相邻元素选择器实现。

7.3 强大的新选择器

CSS 3 新增了很多选择器，它们都很强大，下面为大家详细介绍一下。

:empty 选择器

当没有子元素，并且没有文本节点的时候，:empty 会被触发。利用:empty 给默认提示，代码如下。

```
<style>
  body,div,ul{
    margin:0;
    padding:0;
  }
  .box{
    margin:100px;
  }
  ul{
    margin-top:10px;
  }
  ul:empty::after{
    content:'留言空空如也!';
  }
</style>
<div class="box">
  <input type="text">
  <ul></ul>
</div>
```

在 ul 没有内容的时候，会显示一条默认的信息，如图 7.13 所示。这里用伪类元素“::after”。

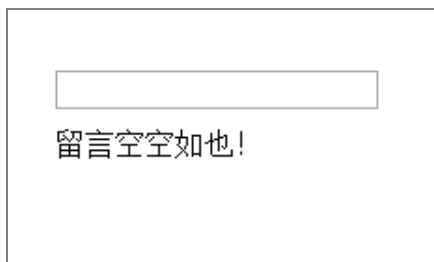


图 7.13 ul 没有内容之前

一旦加上内容，默认信息就会隐藏，如图 7.14 所示。不过需要注意的是，ul 不能换行，必须这样写。如果有换行就匹配不了了，代码如下。

```
<div class="box">
  <input type="text">
  <ul>
    <li>Hello world</li>
    <li>CSS</li>
  </ul>
</div>
```

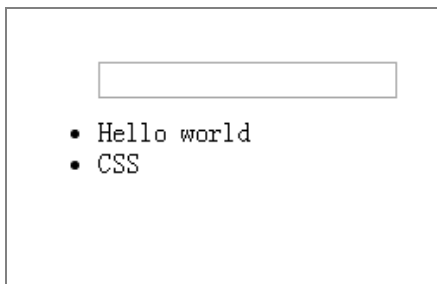


图 7.14 ul 有内容时

购物车也可以利用这个来做默认提示。

:target 选择器

:target 选择器，获取当前描点的那个元素，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  #box:target{
    color:red;
  }
</style>
<a href="#box">点击描点</a>
<div id="box">跳转到我这</div>
```

在默认情况下，如图 7.15 所示。

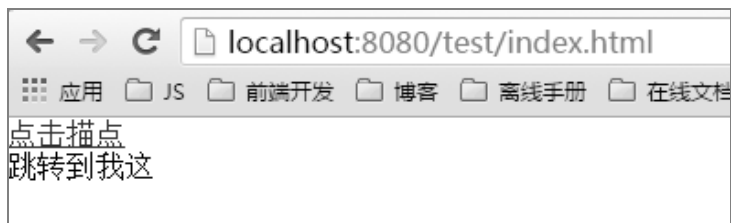


图 7.15 默认情况

单击 a 标签后，如图 7.16 所示。



图 7.16 单击 a 标签后

再实现一个换肤功能，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box > div{
    position:absolute;
    left:0;
    top:0;
    width:100%;
    height:100%;
    z-index:-1;
  }
  #bg1:target{
    background-color:green;
  }
  #bg2:target{
    background-color:red;
  }
}
```

```
#bg3:target{
    background-color:orange;
}
</style>
<div class="box">
    <nav>
        <a href="#bg1">春</a>
        <a href="#bg2">夏</a>
        <a href="#bg3">秋</a>
    </nav>
    <div id="bg1"></div>
    <div id="bg2"></div>
    <div id="bg3"></div>
</div>
```

没单击显示的内容之前，如图 7.17 所示。

单击显示的内容后就可以换肤了。当然，也可以默认给一个背景，而且要给背景元素加“z-index:-1;”，不然定位的元素层级比普通元素高，就单击不了文字了，效果如图 7.18 所示。

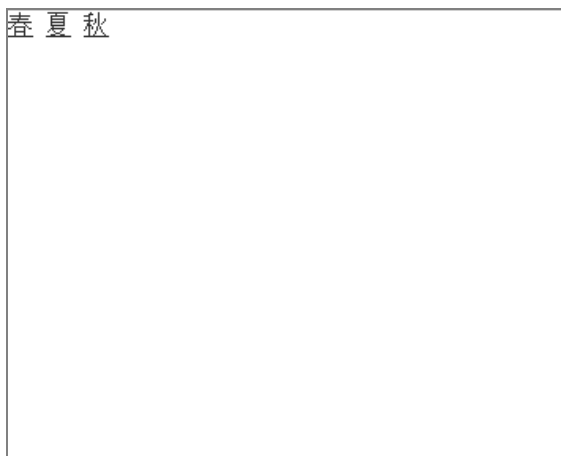


图 7.17 没单击之前

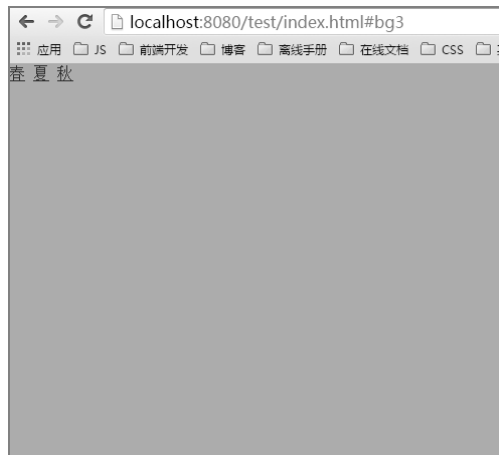


图 7.18 效果图

:target 选择器除了换肤功能，还有很多可以做的，只要是关于单（点）击的，一些简单事情它都可以实现。

(1) 利用:target 选择器实现遮罩层

利用:target 选择器实现遮罩层，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  #show{
    text-decoration:none;
    color:#000;
  }
  #show div{
    position:fixed;
    left:50%;
    top:50%;
    width:200px;
    height:200px;
    transform:translate(-50%,-50%);
    border:10000px solid rgba(0,0,0,0.3);
    display:none;
  }
  #show:target > div{
    display:block;
  }
  #hide:target{
    display:none;
  }
</style>
<div class="box">
  <a href="#show">弹出层</a>
  <a href="#hide" id="show">
    <div id="hide">弹弹弹</div>
  </a>
</div>
```

没单击显示的内容之前，如图 7.19 所示。

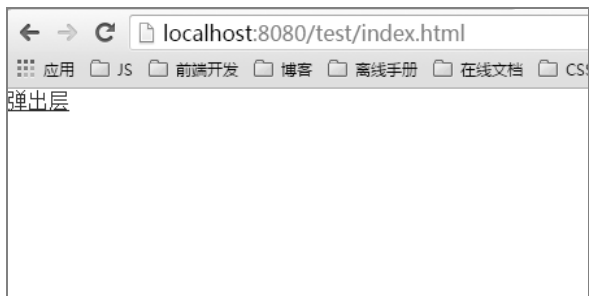


图 7.19 没单击之前

单击显示的内容后，如图 7.20 所示。

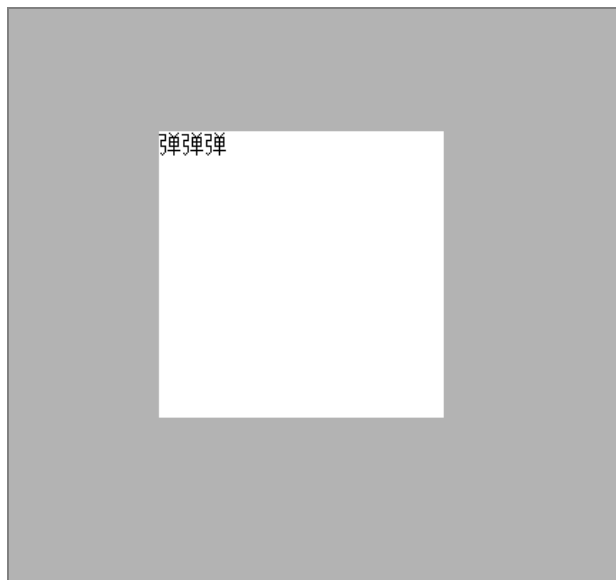


图 7.20 单击之后

当再次单击遮罩层的时候就会关闭，做宣传的时候经常会用到这种效果。此效果的灰色是用 `border` 的边框做的，不然还得用两个标签。不过需要注意的是，这个边框一定要设置大一点，另外里面使用了“`transform:translate(-50%,- 50%);`”，它可以将元素垂直水平居中。代码其实用了两个 `a` 标签，因为这个功能需要做两件事，一件事是显示遮罩层，另一件事是隐藏遮罩层，所以至少得两个 `a` 标签。

(2) 实现 tab 栏切换

实现 tab 栏切换，代码如下。


```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:300px;
    height:300px;
    text-align:center;
    margin:100px auto;
  }
  .title{
    overflow:hidden;
  }
  .title > a{
    float:left;
    width:100px;
    height:35px;
    line-height:35px;
    border:1px solid #dedede;
    box-sizing:border-box;
    text-decoration:none;
  }
  .title > a:not(:first-of-type){
    border-left:none;
  }
  .content{
    position:relative;
  }
  .content > div{
    position:absolute;
    left:0;
    top:0;
    width:100%;
    height:200px;
    line-height:200px;
    border:1px solid #dedede;
    background-color:#fff;
```

```

        border-top:none;
        box-sizing:border-box;
    }
    #content1{
        z-index:1;
    }
    #content1:target,#content2:target,#content3:target{
        z-index:999;
    }
</style>
<div class="box">
    <nav class="title">
        <a href="#content1">CSS</a>
        <a href="#content2">HTML</a>
        <a href="#content3">Javascript</a>
    </nav>
    <div class="content">
        <div id="content1">CSS</div>
        <div id="content2">HTML</div>
        <div id="content3">Javascript</div>
    </div>
</div>

```

效果如图 7.21 所示。



图 7.21 实现 tab 栏切换

其中核心的代码如下。

```
#content1{
    z-index:1;
}
#content1:target,#content2:target,#content3:target{
    z-index:999;
}
```

也就是默认让第一个显示，然后单击的时候让单击的那个元素层级最高。还使用了 `not` 选择器，代码如下。

```
.title > a:not(:first-of-type){
    border-left:none;
}
```

此段代码的意思是，除了第一个 `a` 标签以外，将其他 `a` 标签的 `border-left` 属性设置为 `none`。当然也可以用之前写的“+”或者“~”，效果如图 7.22 所示。



图 7.22 效果图

第 8 章 CSS 图标制作

本章主要讲解如何使用 CSS 制作小图标，并且在只使用一个标签的情况下，不能使用伪元素。在利用 CSS 制作图标时，经常需要用到边框，所以下面先学习一下边框的知识。

8.1 隐藏在边框中的秘密

我们经常使用 `border` 属性来制作一条边框，代码如下。

```
<style>
  .box{
    width:100px;
    height:100px;
    border:1px solid #dedede;
  }
</style>
<div class="box"></div>
```

效果如图 8.1 所示。



图 8.1 经常使用 `border` 属性制作的边框

我们一直这样用着，但它真的长这样吗？不一定。可以给边框加上不同的样式，代码如下，效果如图 8.2 所示。

```
<style>
  .box{
    width:100px;
    height:100px;
    border-top:10px solid red;
    border-right:10px solid green;
    border-bottom:10px solid orange;
    border-left:10px solid pink;
  }
</style>
<div class="box"></div>
```

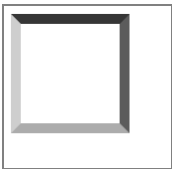


图 8.2 给边框加上不同的样式

每个边框像一个梯形，再把宽度和高度设置为 0，代码如下。

```
.box{
  width:0;
  height:0;
  border-top:20px solid red;
  border-right:20px solid green;
  border-bottom:20px solid orange;
  border-left:20px solid pink;
}
```

效果如图 8.3 所示。



图 8.3 将宽度和高度设置为 0

原来的边框都变成一个三角形，把其他 3 个边框的颜色都设置为透明（透明关键字 transparent），代码如下。

```
.box{
  width:0;
  height:0;
  border-top:20px solid transparent;
  border-right:20px solid transparent;
  border-bottom:20px solid orange;
  border-left:20px solid transparent;
}
<div class="box"></div>
```

效果如图 8.4 所示。



图 8.4 三角形

8.2 边框的烦恼

在使用边框的时候也有很多问题，如图 8.5 所示。



图 8.5 边框多出 1px

后面两个边框的左边会多 1px，出现这个问题的原因就是，前一个元素的右边框和后面一个元素的左边框碰到一起了。解决这个问题的办法就是把后面元素的左边框去掉，或者设置 margin 为负值。

除第一个元素以外，其他元素去掉左边框，代码如下。

```
<style>
```

```

nav a{
    float:left;
    width:100px;
    height:35px;
    line-height:35px;
    margin:100px auto;
    text-align:center;
    text-decoration:none;
    border:1px solid #dedede;
}
nav a ~ a{
    border-left:none;
}
</style>
<nav>
    <a href="">CSS</a>
    <a href="">HTML</a>
    <a href="">Javascript</a>
</nav>

```

效果如图 8.6 所示。



图 8.6 其他元素去掉左边框

除第一个元素以外，其他元素的 `margin-left` 都设置为 `-1px`，代码如下。

```

<style>
nav a{
    float:left;
    width:100px;
    height:35px;
    line-height:35px;
    margin:100px auto;
    text-align:center;
    text-decoration:none;
    border:1px solid #dedede;

```

```

    }
    nav a ~ a{
        margin-left:-1px;
    }
</style>
<nav>
    <a href="">CSS</a>
    <a href="">HTML</a>
    <a href="">Javascript</a>
</nav>

```

效果如图 8.7 所示。



图 8.7 设置 margin-left 为负值

设置 margin 为负值的原理就是，让前一个元素的右边框和后一个元素的左边框重合。

还有一个问题，即如果一个元素一开始没有边框，而是后面动态加上去的，那么它就会抖动，代码如下。

```

<style>
    .box{
        width:100px;
        height:100px;
        line-height:100px;
        margin:100px auto;
        text-align:center;
        background-color:pink;
    }
    .box:hover{
        border:10px solid #dedede;
    }
</style>
<div class="box">box</div>

```

出现这个问题的根本原因是，一开始没有边框，后面突然加上去时边框把空间撑大了，所以内容的位置就可能改变，自然就会抖动一下。要解决这个问题很简单，给 box 加一个

透明的边框，代码如下。

```
.box{
  width:100px;
  height:100px;
  line-height:100px;
  margin:100px auto;
  text-align:center;
  background-color:pink;
  border:10px solid transparent;
}
```

原理是，将边框颜色设置成透明的，自然就看不出边框了。或者在没有背景色的情况下，给 box 一个 padding，当 box 设置为 hover 的时候再去掉 padding，代码如下。

```
<style>
  .box{
    width:100px;
    height:100px;
    line-height:100px;
    margin:100px auto;
    text-align:center;
    padding:10px;
  }
  .box:hover{
    padding:0;
    border:10px solid #dedede;
  }
</style>
```

这么做的前提是 box 没有背景色，不然 padding 也是会应用背景的。

8.3 边框的孪生兄弟——outline

有一个与边框很像的属性——outline，可以用来绘制元素的轮廓，其属性和 border 几乎一模一样，不过 border-radius 并没有作用在 outline 上，与 border 最大的区别就是 outline 不会占据空间，代码如下。

```
<style>
```

```

.box{
  width:100px;
  height:100px;
  line-height:100px;
  text-align:center;
  outline:1px solid #dedede;
}
</style>
<div class="box">outline</div>

```

效果如图 8.8 所示。

可以看到它和 border 好像一模一样,来看看它的大小,如图 8.9 所示。而如果是用 border,则大小如图 8.10 所示。

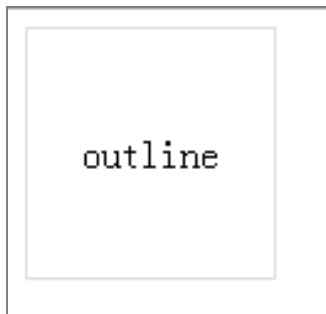


图 8.8 outline

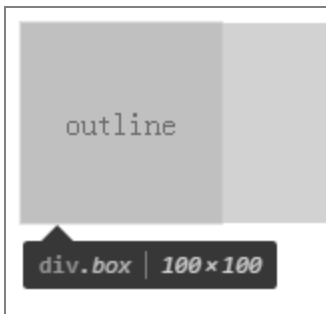


图 8.9 使用 outline 的效果



图 8.10 使用 border 的效果

使用 outline 的效果和使用“box-sizing:border-box;”的效果非常像,代码如下。

```

<style>
.box{
  width:100px;
  height:100px;
  line-height:100px;
  text-align:center;
  border:1px solid #dedede;
  box-sizing:border-box;
}
</style>

```

效果如图 8.11 所示。

可以看到,两个效果一模一样,只不过是后面一种添加一个 box-sizing: border-box;属

性。如果不想增加边框宽度，那么还是建议使用 `outline`，使用 `outline` 可以很好地解决前面讲的用 `border` 的抖动问题，代码如下。

```
<style>
  .box{
    width:100px;
    height:100px;
    line-height:100px;
    margin:100px auto;
    text-align:center;
    background-color:pink;
  }
  .box:hover{
    outline:5px solid #dedede;
  }
</style>
<div class="box">outline</div>
```

效果如图 8.12 所示。

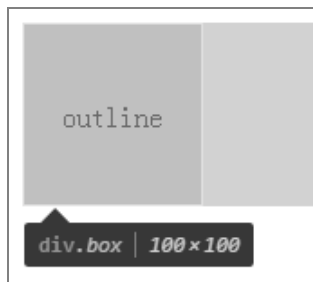


图 8.11 使用 box-sizing 的效果

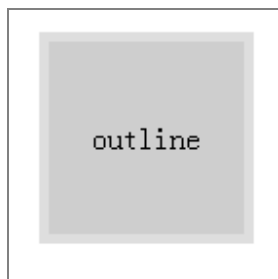


图 8.12 outline 解决抖动问题

在一开始提到了 `border-radius` 不会作用在 `outline` 上，代码如下。

```
<style>
  .box{
    width:100px;
    height:100px;
    line-height:100px;
    margin:100px auto;
    text-align:center;
    background-color:pink;
  }
```

```

        outline:5px solid #dedede;
        border-radius:20px;
    }
</style>
<div class="box">outline</div>

```

效果如图 8.13 所示。



图 8.13 border-radius 不会作用在 outline 上

可以看到 border-radius 并没有作用在 outline 上，另外 outline 也不占据空间，代码如下。

```

<style>
    .box{
        width:100px;
        height:100px;
        line-height:100px;
        text-align:center;
        background-color:pink;
        outline:50px solid #dedede;
        border-radius:20px;
    }
</style>
<div class="box">outline</div>
<p>我是来捣乱的!!!!!!!!!!!! </p>

```

效果如图 8.14 所示。

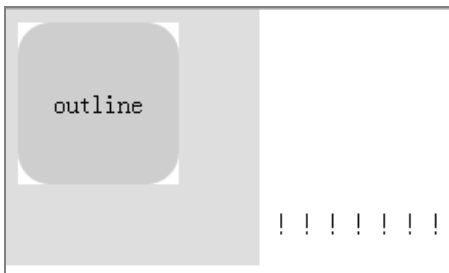


图 8.14 outline 不占据空间

可以看到它把 p 标签的内容盖住了，说明它确实是不占据空间的。如果想不占空间，又想使用 border-radius 怎么办呢？那就先使用“border+border-radius”，再利用 box-sizing 属性来解决，代码如下。

```
<style>
  .box{
    width:100px;
    height:100px;
    line-height:100px;
    text-align:center;
    background-color:pink;
    border:10px solid #dedede;
    border-radius:20px;
    box-sizing:border-box;
  }
</style>
<div class="box">outline</div>
```

效果如图 8.15 所示。除了这个方法之外，还有一个属性也可以完成同样的事，并且也是不占空间的。它就是 box-shadow 属性，代码如下。

```
<style>
  .box{
    width:100px;
    height:100px;
    line-height:100px;
    margin:100px;
    text-align:center;
    background-color:pink;
    box-shadow:0 0 0 10px #dedede;
```

```
border-radius:10px;
}
</style>
<div class="box">box-shadow</div>
```

效果如图 8.16 所示。

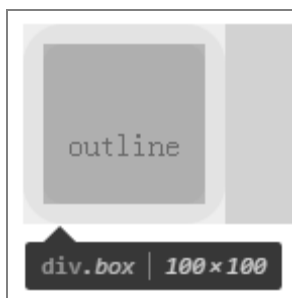


图 8.15 使用 box-sizing 属性

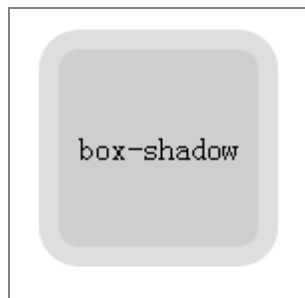


图 8.16 使用 box-shadow 属性

box-shadow 语法如下：

box-shadow:x 轴偏移量 y 轴偏移量 阴影大小 边框大小 填充的颜色（可选）

虽然用 box-shadow 的第 4 个参数边框大小来模拟边框也可以实现我们要的效果，但其实是有缺陷的，即不能使用边框的一些属性。因为它根本就不是边框，所以还是 border 最适合，然后加上 box-sizing。当然，如果不需要使用边框的其他样式，这个属性还是有用的。其实也可以使用 calc 函数来达到我们的目的，用宽度减去边框的大小即可。

- 当动态地添加边框的时候会抖动；
- 结合边框的特性可以制作很多小图标；
- outline 属性和 border 属性的区别就是它不会占位置，缺点是不能使用圆角属性；
- 利用 box-shadow 可以简单地模拟 outline 属性。

8.4 纯 CSS 图标制作

1. 平行四边形图标

使用一个 skew 函数即可实现平行四边形图标的制作，代码如下。

```
<style>
.box{
```

```

        width:50px;
        height:50px;
        margin-left:100px;
        background-color:red;
        transform:skew(-25deg);
    }
</style>
<div class="box"></div>

```

平行四边形图标如图 8.17 所示。

2. 暂停按钮

暂停按钮如图 8.18 所示。



图 8.17 平行四边形图标

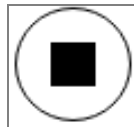


图 8.18 暂停按钮

代码如下。

```

<style>
    .box{
        width:50px;
        height:50px;
        color:#000;
        border: 1px solid;
        border-radius: 100%;
        outline: 10px solid;
        outline-offset: -26px;
        cursor:pointer;
    }
</style>
<div class="box"></div>

```

暂停按钮实现的原理就是边框用 **border**，里面的正方形用 **outline**。因为 **outline** 有一个 **offset** 属性可以用来设置偏移量，并且是按照比例来的。因此制作这样的—个效果就比较简单了。其实如果再将 **outline-offset** 的值设置小一点，一个加号就出来了，效果如图 8.19 所示。

将 `outline-offset` 改为 `-35px` 即可，如果再将其旋转，就变成了一个关闭按钮，效果如图 8.20 所示。

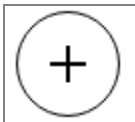


图 8.19 加号



图 8.20 关闭按钮

代码如下。

```
<style>
  .box{
    width: 50px;
    height: 50px;
    background-color: #000;
    border-radius: 100%;
    outline: 15px solid #fff;
    outline-offset: -39px;
    transform: rotate(45deg);
    cursor: pointer;
  }
</style>
```

3. 制作汉堡菜单

汉堡菜单如图 8.21 所示。



图 8.21 汉堡菜单

代码如下。

```
<style>
  .box{
    width: 50px;
    height: 0px;
    color: red;
    box-shadow: 36px 10px 0px 3px, 36px 0px 0px 3px, 36px 20px 0px 3px;
  }
</style>
```



```
</style>
<div class="box"></div>
```

实现汉堡菜单主要是利用 `box-shadow` 可以使用多个值，并且可以设置偏移量。其使用背景裁剪属性也可以做出来，代码如下。

```
<style>
  .box{
    width: 30px;
    height: 3px;
    padding: 2px 0;
    border-top: 3px solid red;
    border-bottom: 3px solid red;
    background-clip: content-box;
    background-color: red;
  }
</style>
```

间隔用 `padding` 属性，但是默认背景是包括 `padding` 的，因此使用背景裁剪属性 `background-clip`，这样背景就不包括 `padding` 值了，效果如图 8.22 所示。

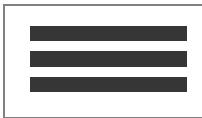


图 8.22 利用背景裁剪属性

当然，还可以使用渐变函数来制作这个效果，代码如下。

```
<style>
  .box{
    width:15px;
    height:12px;
    background:linear-gradient(to bottom,red 0%,red 20%,transparent
20%,transparent 40%,red 40%,red 60%,transparent 60%,transparent 80%,red 80%,red
100%);
  }
</style>
```

原理就是每隔一段创建一个透明色。

其实只需要制作一个长方形，然后用 `background-size` 属性将图片设置小一点，背景就会进行平铺，代码如下。

```

<style>
  .box{
    width: 15px;
    height: 15px;
    background: linear-gradient(to bottom,red 50%,transparent 50%);
    background-size: 5px;
  }
</style>

```

效果如图 8.23 所示。如果再加一个外轮廓，就变成了一个文章图标，如图 8.24 所示。



图 8.23 利用 background-size 属性



图 8.24 文章图标

代码如下。

```

<style>
  .box{
    width: 15px;
    height: 15px;
    background: linear-gradient(to bottom,red 50%,transparent 50%);
    background-size:5px;
    outline: 1px solid red;
    outline-offset: 4px;
  }
</style>

```

这里不用边框是因为边框是靠着内容区的，虽然外轮廓也是，但它有一个偏移属性，因此用 outline 属性比较合适。

4. 单选按钮

单选按钮如图 8.25 所示。

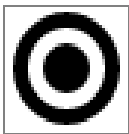


图 8.25 单选按钮

代码如下。

```
<style>
  .box{
    width: 16px;
    height: 16px;
    background-color: #000;
    border-radius: 100%;
    box-shadow: 0px 0px 0px 5px #fff,0px 0px 0px 10px #000;
    cursor: pointer;
  }
</style>
<div class="box"></div>
```

因为 box-shadow 会按比例缩放，因此将第一个值设置为白色，然后将第二个值设置的比第一个值大就可以了。将代码修改一下，就可以做出如图 8.26 所示的效果。



图 8.26 圆圈里面带个十字

代码如下。

```
<style>
  .box{
    width: 14px;
    height: 14px;
    background-color: #000;
    border-radius: 100%;
    box-shadow: 0px 0px 0px 3px #fff,0px 0px 0px 5px #000;
    outline: 19px solid #fff;
    outline-offset: -25px;
    transform: scale(1.5);
    cursor: pointer;
    margin: 100px;
  }
</style>
```

由于不太好调整，因此里面使用了缩放函数 scale，现在显示的就是一些效果的结合。

这个单选按钮也可以使用背景裁剪属性来做，代码如下。

```
<style>
  .box{
    width: 15px;
    height: 15px;
    padding:7px;
    border: 3px solid green;
    border-radius:100%;
    background-clip: content-box;
    background-color: green;
  }
</style>
```

效果如图 8.27 所示。

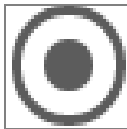


图 8.27 利用背景裁剪属性做单选按钮

5. 田型图标

田型图标如图 8.28 所示。



图 8.28 田型图标

代码如下。

```
<style>
  .box{
    width: 0px;
    color: red;
    border: 3px solid;
    outline: 6px dotted;
    outline-offset: 6px;
    margin: 100px;
  }
```

```
</style>
<div class="box"></div>
```

外面的几个正方形用外轮廓实现，正中间的正方形通过边框实现。也可以通过 padding 来实现，代码如下。

```
<style>
  .box{
    width: 0px;
    padding: 3px;
    background-color: red;
    outline: 6px dotted red;
    outline-offset: 6px;
    margin: 100px;
  }
</style>
```

当然也可以直接设置宽度和高度来实现，代码如下。

```
<style>
  .box{
    width: 6px;
    height: 6px;
    background-color: red;
    outline: 6px dotted red;
    outline-offset: 6px;
    margin: 100px;
  }
</style>
```

6. 下载箭头

下载箭头如图 8.29 所示。



图 8.29 下载箭头

代码如下。

```
<style>
  .box{
    width: 0;
```

```

        color:red;
        border: 8px solid transparent;
        border-top: 8px solid;
        box-shadow: 0px -12px 0 -4px;
    }
</style>

```

使用 `border` 制作三角形，使用 `box-shadow` 制作正方形，主要用了偏移，直接用宽度和高度制作正方形是不行的，因为三角形和正方形始终是一样大的，下面再进行扩展，效果如图 8.30 所示。

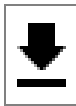


图 8.30 三角形带下划线

代码如下。

```

<style>
    .box{
        width: 1px;
        height: 6px;
        color: #000;
        border: 8px solid transparent;
        border-top: 8px solid;
        box-shadow: 0px -12px 0 -4px;
        background: linear-gradient(to bottom,#fff 50%,#000 50%) repeat-x;
    }
</style>

```

底下的长方形用渐变函数来写，然后水平方向平铺，主要是因为宽度只有 `1px`，而且宽度不能设置太大，否则三角形就会变形。

7. 书签图标

书签图标如图 8.31 所示。



图 8.31 书签图标

代码如下。

```
<style>
  .box{
    width: 0;
    height: 8px;
    background-color: orange;
    border: 8px solid transparent;
    border-bottom: 8px solid #fff;
  }
</style>
```

实现这种效果的原理就是将三角形设置成背景色，这样空心的三角形就出现了。

8. 两个半圆图标

两个半圆图标如图 8.32 所示。



图 8.32 两个半圆图标

代码如下。

```
<style>
  .box{
    width: 50px;
    height: 50px;
    border-radius: 100%;
    background: linear-gradient(to right, #ccc 50%, #000 50%);
  }
</style>
```

这个比较简单，就是通过渐变函数实现，然后来个圆角边框。

9. 禁用图标

禁用图标如图 8.33 所示。



图 8.33 禁用图标

代码如下。

```
<style>
    .box{
        width: 20px;
        height: 20px;
        border-radius: 100%;
        border: 2px solid #000;
        background: linear-gradient(to right,#fff 45%,#000 45%,#000 55%,#fff
55%);
        transform: rotate(40deg);
    }
</style>
```

外圈利用圆角边框，里面的竖线用渐变来做，然后再用旋转属性即可。

10. 左右箭头图标

左右箭头图标如图 8.34 所示。



图 8.34 左右箭头图标

既然能做出一个三角形，那么就可以做出两个三角形。不过不需要重新添加一个元素来实现两个三角形，这里可以使用 CSS 3 中的投影属性，代码如下。

```
<style>
    .box{
        width: 0px;
        height: 0px;
        border: 10px solid transparent;
        border-left: 10px solid red;
        -webkit-box-reflect: left 5px;
        box-reflect: left 5px;
    }
</style>
```

需要在 Chrome 浏览器中打开，因为其他浏览器或许不支持。

11. 梯形图标

梯形图标如图 8.35 所示。



图 8.35 梯形图标

代码如下。

```
<style>
    .box{
        width: 50px;
        border: 50px solid transparent;
        border-bottom: 50px solid red;
    }
</style>
```

在三角形的基础上设置宽度就可以了。

12. 鹰嘴图标

鹰嘴图标如图 8.36 所示。



图 8.36 鹰嘴图标

代码如下。

```
<style>
    .box{
        width: 35px;
        border-top: 50px solid transparent;
        border-right: 22px solid #096;
        border-bottom-right-radius: 100%;
    }
</style>
```



第 9 章 你今天换背景了吗

9.1 对背景属性的深入探索

在页面中难免需要使用背景，举一个例子，代码如下。

```
<style>
  .box{
    width:100px;
    height:100px;
    background-color:pink;
  }
</style>
<div class="box">一只会唱歌的鸟</div>
```

这是一种常见的情况，如果加上 padding，则 padding 也会应用背景颜色，代码如下。

```
.box{
  width:100px;
  height:100px;
  padding:50px;
  background-color:pink;
}
```

效果如图 9.1 所示。

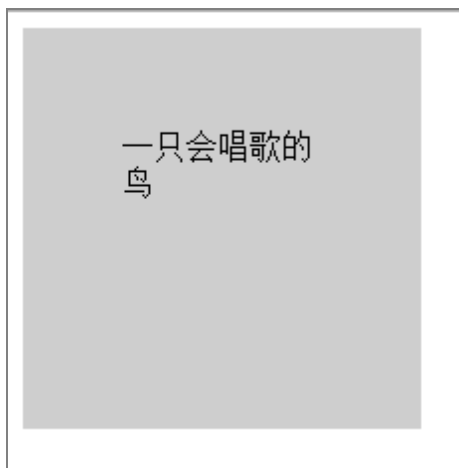


图 9.1 padding 也会应用背景颜色

当父元素设置背景色，而子元素没有设置背景色时，父元素的背景会充当子元素的背景，代码如下。

```
<style>
  body{
    background-color:red;
  }
  .box{
    width:100px;
    height:100px;
    margin:50px;
  }
</style>
```

效果如图 9.2 所示。

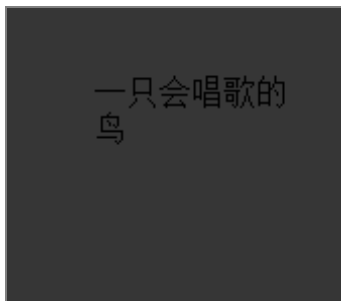


图 9.2 父元素的背景充当子元素的背景

经常会用这个特点来简化网页的开发。比如一个页面中大部分的背景颜色都是一样的，那么可以给 `body` 添加背景颜色，若不一样再单独设置。

除使用背景色以外，有时也会使用背景图片，代码如下。

```
<style>
  .box{
    width:500px;
    height:500px;
    background:url(images/4.jpg);
  }
</style>
<div class="box"></div>
```

效果如图 9.3 所示。



图 9.3 使用背景图片

默认图片的原点是在左上角，如果想改变图片的位置可以修改代码，代码如下。

```
<style>
  .box{
```

```
width:500px;
height:500px;
background:url(images/4.jpg) 50% 50%;
}
</style>
<div class="box"></div>
```

效果如图 9.4 所示。



图 9.4 改变背景图片位置

background 属性语法如下：

```
background:color url repeat x y repeat;
```

x 、 y 单位可以是 px、%、em、ch 等。 x 的单位还可以是 left、center、right。 y 的单位还可以是 top、center、bottom。

如果是使用 px、em、ch 等单位，就要把图片往左移，得用负值，代码如下。

```
.box{
width:500px;
```

```
height:500px;  
background:url(images/4.jpg) -100px;  
}
```

效果如图 9.5 所示。



图 9.5 使用负值的图片效果

另外一点就是，如果只写一个值，那么 y 轴的值将是 center 或者说是 50%。如果使用 top、right、bottom、left 或者百分号，那么值在默认情况下，x 轴是往左移动的，y 轴是往上移动的，也就是不需要写负值，代码如下。

```
<style>  
.box{  
width:500px;  
height:500px;  
background:url(images/4.jpg) 100% 100%;  
}  
</style>
```

效果如图 9.6 所示。

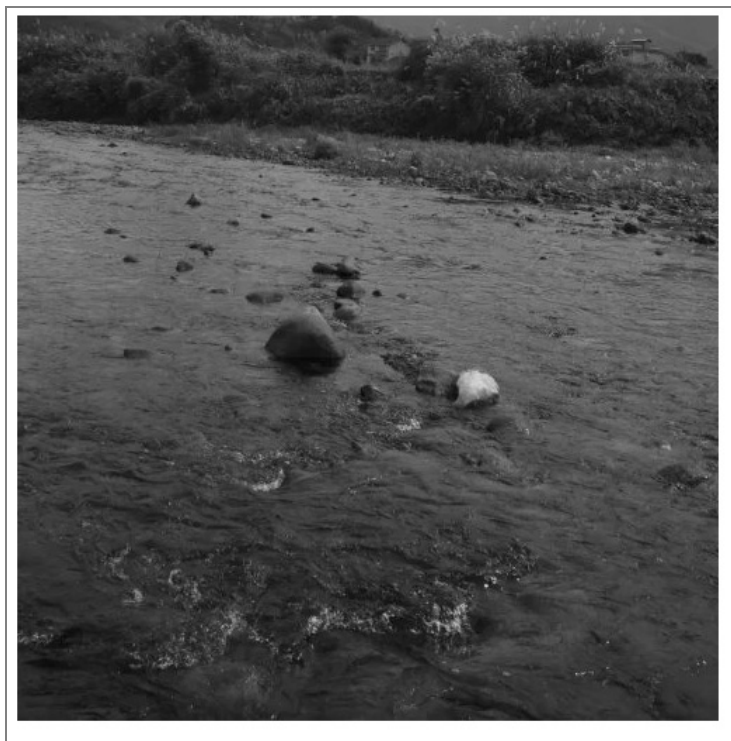


图 9.6 百分比不写负值的图片效果

因为两个值都是 100%，所以有些人就想着可以简写了，但是，如果只写一个值，后一个值默认是 50%，所以除非第二个正好是想呈现设置为 50% 的效果，否则不要这么设置。

9.2 新增的背景功能

CSS 3 对背景属性进行了扩展，其中有很多强大的属性，比如 `background-origin`，它可以用来改变背景的原点；`background-size` 还可以用来改变背景的大小。

9.2.1 改变背景原点——`background-origin`

`background-origin` 可以改变背景的原点。它有以下 3 个属性。

- padding-box: 从 padding 区域（含 padding）开始显示背景图像；
- border-box: 从 border 区域（含 border）开始显示背景图像；
- content-box: 从 content 区域开始显示背景图像。

举一个例子，代码如下，效果如图 9.7 所示。

```
<style>
  div{
    float:left;
    width:100px;
    height:100px;
    padding:20px;
    margin-right:10px;
    border:20px dotted pink;
    background:url(images/4.jpg) no-repeat;
  }
  div:nth-of-type(2){
    background-origin:border-box;
  }
  div:nth-of-type(3){
    background-origin:padding-box;
  }
  div:nth-of-type(4){
    background-origin:content-box;
  }
</style>
<div>默认</div>
<div>border</div>
<div>padding</div>
<div>content</div>
```

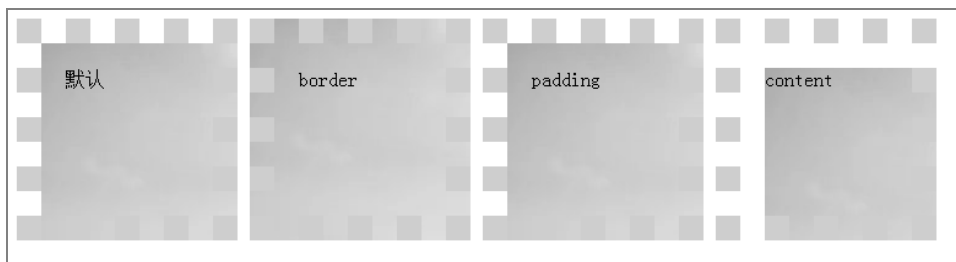


图 9.7 改变背景原点

需要注意的是，如果背景 `background-attachment` 设置为 `fixed`，这个属性将不起作用。默认值是 `padding-box`。

9.2.2 背景裁剪——`background-clip`

`background-clip` 拥有以下 4 个属性。

- `padding-box`：从 `padding` 区域（不含 `padding`）开始向外裁剪背景；
- `border-box`：从 `border` 区域（不含 `border`）开始向外裁剪背景；
- `content-box`：从 `content` 区域开始向外裁剪背景；
- `text`：从前景内容的形状（比如文字）作为裁剪区域向外裁剪。

举一个例子，代码如下，效果如图 9.8 所示。

```
<style>
  div{
    float:left;
    width:100px;
    height:100px;
    padding:20px;
    margin-right:10px;
    border:20px dotted pink;
    background-color:red;
  }
  div:nth-of-type(2){
    background-clip:border-box;
  }
  div:nth-of-type(3){
    background-clip:padding-box;
  }
  div:nth-of-type(4){
    background-clip:content-box;
  }
  div:nth-of-type(5){
    -webkit-background-clip:text;
  }
</style>
<div>默认</div>
<div>border</div>
```

```

<div>padding</div>
<div>content</div>
<div>text</div>

```

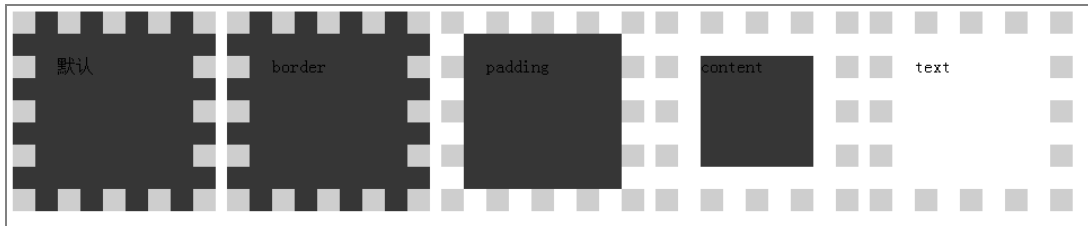


图 9.8 背景裁剪

需要注意的是，text 需要加-webkit-前缀。

这里的裁剪是说对背景进行裁剪，不要认为使用了“background-clip: border-box;”就会把边框也裁剪了。就算把所有内容都裁剪了，也不会影响元素，background-clip 属性中的名称就说明了是背景裁剪。

1. 利用 background-clip 制作一个图片呈缩放效果

利用 background-clip 制作一个图片呈缩放效果代码如下。

```

<style>
  .box{
    width:100px;
    height:100px;
    padding:10px;
    border:1px solid #dedede;
    background:url(images/1.jpg);
    background-clip:content-box;
  }
  .box:hover{
    background-clip:padding-box;
  }
</style>
<div class="box"></div>

```

在默认情况下效果如图 9.9 所示。当鼠标移动到这个 div 时，如图 9.10 所示。



图 9.9 默认情况下



图 9.10 当鼠标移动到这个 div 时

举一个例子，拿到一个图，想做成如图 9.11 所示的效果。但是做出来的效果如图 9.12 所示。



图 9.11 想做出的效果图

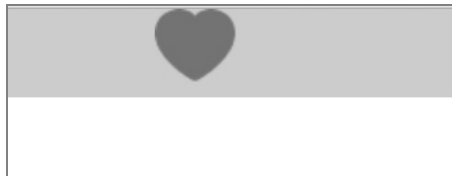


图 9.12 实际做出的效果图

现在想让中间的爱心在 y 轴方向向下移动一点，给 p 标签加上 “padding-top: 10px;”，如图 9.13 所示。

所显示的红心并没有向下移动，而是高度多出了一些。之所以出现这样的情况，是因为背景包括了 padding。但是现在并不想呈现这样的效果，这时就可以用 “background-clip: content-box”（从内容开始计算背景），那么就不会包括 padding 了，效果如图 9.14 所示。

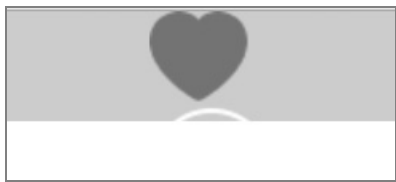


图 9.13 用 padding-top 后的结果



图 9.14 运用了 background-clip: content-box; 的效果

除了 text，其他的几个属性都好理解。其实只设置 text 属性没多大用，主要原因是它

的功能就是把背景应用到文字上，而默认文字是实色的，所以看不出什么效果。不过，如果使用 `-webkit-text-fill-color` 属性，效果就很明显了。这个属性是用来设置文字填充色的，代码如下。

```
<style>
  div{
    font-size:100px;
    -webkit-text-fill-color: transparent;
    background:url(images/1.jpg) 50%;
    -webkit-background-clip:text;
  }
</style>
<div>看不到我，看不到我</div>
```

效果如图 9.15 所示。



图 9.15 使用 text 属性的效果图

将文字填充色改成透明的，就可以看到使用“`background-clip:text;`”的效果了。还可以利用它制作渐变字体，代码如下。

```
<style>
  div{
    font-size:100px;
    -webkit-text-fill-color: transparent;
    background:linear-gradient(to bottom,#ccc,#fff);
    -webkit-background-clip:text;
  }
</style>
<div>看不到我，看不到我</div>
```

效果如图 9.16 所示。



图 9.16 制作渐变字体

9.2.3 设置背景图片大小——background-size

background-size 用来设置背景图像的尺寸大小，它有以下 5 个属性。

- length: 用长度值指定背景图像大小，不允许为负值；
- %: 用百分比指定背景图像大小，不允许为负值；
- auto: 背景图像的真实大小；
- cover: 将背景图像等比缩放到完全覆盖容器，背景图像有可能超出容器；
- contain: 将背景图像等比缩放到宽度或高度与容器的宽度或高度相等，背景图像始终被包含在容器内。

该属性支持传递两个参数（如果值为 cover 或 contain 则不支持传递两个）：第一个用于定义背景图像的宽度；第二个用于定义背景图像的高度；如果只提供一个参数值，该值将用于定义背景图像的宽度，第 2 个值默认为 auto，即高度为 auto。此时背景图以提供的宽度作为参照来进行等比缩放。

代码如下。

```
<style>
    div{
        width:200px;
        height:200px;
        outline:1px solid #dedede;
        margin-right:5px;
        float:left;
        background:url(images/2.jpg) no-repeat;
    }
    div:nth-of-type(1){
        background-size:auto;
    }
    div:nth-of-type(2){
        background-size:cover;
    }
    div:nth-of-type(3){
        background-size:contain;
    }
    div:nth-of-type(4){
        background-size:100% 100%;
    }
}
```

```
</style>  
<div>auto</div>  
<div>cover</div>  
<div>contain</div>  
<div>100%</div>
```

效果如图 9.17 所示。



图 9.17 综合例子

如果想更加精确地控制背景大小，还得使用单位。

从 CSS3 开始，一个元素可以设置多个背景，如图 9.18 所示。



图 9.18 设置多个背景

图 9.18 所示的效果就可以用多重背景来实现，代码如下。

```
<style>  
div{  
width:500px;
```

```
height:300px;  
background:url(images/5.png) no-repeat center,url(images/6.png) no-repeat  
center;  
}  
</style>  
<div></div>
```

效果如图 9.19 所示。



图 9.19 多重背景

多个背景用逗号隔开，涉及多重背景时就可以使用这个属性。

9.3 总结

- 背景默认原点是在左上角，如果想改变原点的位置，可以使用 `background-origin` 属性；
- 背景还有一个裁剪属性，可以按照某个原点位置裁剪掉；
- 背景定位如果只写一个值，后一个默认为居中。



第 10 章 让文字更美一些

你是否想过使用 CSS 制作精美的文字效果呢？

文字阴影效果，如图 10.1 所示。



图 10.1 文字阴影效果

文字模糊效果，如图 10.2 所示。



图 10.2 文字模糊效果

文字镂空效果，如图 10.3 所示。



图 10.3 文字镂空效果

渐变文字效果，如图 10.4 和图 10.5 所示。



图 10.4 渐变文字效果（1）



图 10.5 渐变文字效果（2）

10.1 制作非主流文字

使用 `text-shadow` 制作文字阴影的效果，也可以使用 CSS 滤镜属性来制作。

`text-shadow` 的语法如下：

```
text-shadow:x,y,shadowRadius,color;
```

里面支持传递 4 个参数，即 x 轴偏移量、 y 轴偏移量、阴影模糊值（可选）、阴影颜色（可选）。举一个例子，代码如下。

```
<style>
  div{
    font-size: 60px;
    text-shadow:5px 20px 3px red;
  }
</style>
<div>一场雨。</div>
```

效果如图 10.6 所示。

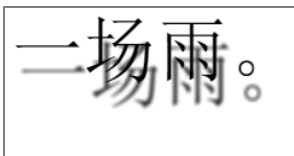


图 10.6 演示效果

文字模糊效果制作的方法比较多，这里举几个常用的方法。

用 `rgba` 制作文字模糊度，代码如下。

```
div{
  font-size: 60px;
  color:rgba(0,0,0,0.3);
}
```

用 `opacity` 制作文字模糊度，代码如下。

```
div{
  font-size: 60px;
  opacity:0.3;
}
```

用 `opacity` 制作文字模糊度的效果，如图 10.7 所示。



图 10.7 用 opacity 制作文字模糊度的效果

或许你会说模糊的效果有些牵强，看上去像文字阴影的效果。那再来看看使用阴影模糊度，代码如下。

```
div{  
  font-size: 60px;  
  text-shadow: 0 0 10px;  
}
```

效果如图 10.8 所示。



图 10.8 使用文字阴影模糊度的效果

这个模糊是给阴影的，而不影响文字本身。前面讲的 `text-fill-color` 属性是用来设置文字填充色的，如果将文字变成透明的，那是不是就只剩下文字阴影了呢？代码如下。

```
div{  
  font-size: 60px;  
  text-shadow: 0 0 8px;  
  -webkit-text-fill-color: transparent;  
}
```

效果如图 10.9 所示。

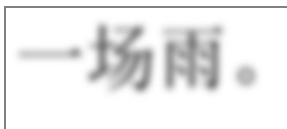


图 10.9 将文字变成透明的效果

如果想制作文字镂空的效果，需要配合文字描边属性——`text-stroke`，目前只有 webkit 内核支持，代码如下。

```
div{  
  font-size: 60px;  
  text-stroke: 2px solid black;
```

```

-webkit-text-fill-color:transparent;
-webkit-text-stroke:1px red;
}

```

文字镂空效果，如图 10.10 所示。



图 10.10 文字镂空效果

`font-size` 参数表示描边大小；`-webkit-text-fill-color` 参数表示描边颜色。加上文字阴影效果，代码如下。

```

div{
  font-size: 60px;
  -webkit-text-fill-color:transparent;
  -webkit-text-stroke:1px red;
  text-shadow: 0 0 63px red;
}

```

结合文字阴影的效果，如图 10.11 所示。



图 10.11 结合文字阴影的效果

如果制作文字渐变效果，就需要使用渐变属性，但好像没有文字渐变这个属性。那么可不可以“创建”一个渐变属性呢？当然可以，代码如下。

```

div{
  font-size: 60px;
  -webkit-text-fill-color:transparent;
  background: linear-gradient(to bottom,#fff,red);
  -webkit-background-clip: text;
}

```

文字渐变效果，如图 10.12 所示。



图 10.12 文字渐变效果

需要注意的是，需要将填充颜色制作成透明的，不然就看不出效果了。

10.2 新增的文字对齐属性

在 CSS2 中，文字对齐属性比较单调，很难满足项目需求。如果想实现更多的文字对齐属性，只能结合一些其他的技巧来实现。而在 CSS 3 中新增了一些文字对齐的属性。

10.2.1 文字两端对齐

常用的 `text-align` 新增了以下 4 个属性。

- `justify`: 内容两端对齐，但对于换行及最后一行（包括仅有一行文本的情况，因为它既是第一行也是最后一行）不做处理；
- `start`: 如果是水平方向，则类似于 `left`；如果是垂直方向，则类似于 `top`；
- `end`: 如果是水平方向，则类似于 `right`；如果是垂直方向，则类似于 `bottom`；
- `justify-all`: 效果等同于 `justify`，但会对最后一行也两端对齐（Chrome 浏览器不支持此属性）。

举个例子，代码如下。

```
<style>
  div{
    width:300px;
    border:1px solid #dedede;
  }
  div:nth-of-type(1){
    text-align:justify;
  }
  div:nth-of-type(1)::after{
    content:'';
    display:inline-block;
    width:100%;
```

```

        visibility:hidden;
    }
    div:nth-of-type(2){
        text-align:start;
    }
    div:nth-of-type(3){
        text-align:end;
    }
    div:nth-of-type(4){
        text-align:justify-all;
    }
</style>
<div>这是 justify</div>
<div>这是 start</div>
<div>这是 end</div>
<div>这是 justify-all</div>

```

效果如图 10.13 所示。

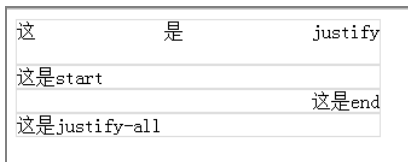


图 10.13 综合例子

因为 `justify` 不会对第一行起作用，所以这里使用了伪元素让它强制换行，这样它才能起作用。或者文本是多行的，总之就是它不会对最后一行起作用。另外，`justify-all` 在浏览器中没有任何效果，如果看到它和第一行的效果一样，那么证明浏览器支持这个属性。另外，如果想让 `start` 和 `end` 在垂直方向起作用，还需要配合一些其他的属性，比如书写模式 `writing-mode`。举个例子，代码如下。

```

<style>
    div{
        width:100px;
        height:200px;
        float:left;
        writing-mode:vertical-lr;
        border:1px solid #dedede;
    }

```

```

    }
    div:nth-of-type(1){
        text-align:start;
    }
    div:nth-of-type(2){
        text-align:end;
    }
</style>
<div>这是 start</div>
<div>这是 end</div>

```

效果如图 10.14 所示。



图 10.14 结合书写模式

10.2.2 末尾文本对齐

和 `text-align` 属性很像的一个属性——`text-align-last`，它是用来设置最后一行文字的对齐方式，与 `text-align` 的属性几乎一模一样。只不过 `text-align` 不包含最后一行，而 `text-align-last` 则相反。举个例子，代码如下。

```

<style>
    div{
        width:150px;
        float:left;
        margin-right:5px;
        border:1px solid #dedede;
    }
    div:nth-of-type(1){

```

```

    text-align-last:justify;
  }
  div:nth-of-type(2){
    text-align-last:start;
  }
  div:nth-of-type(3){
    text-align-last:end;
  }
</style>
<div>这是一段很长很长的文字</div>
<div>这是一段很长很长的文字</div>
<div>这是一段很长很长的文字</div>

```

效果如图 10.15 所示。

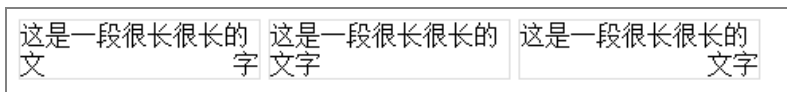


图 10.15 末尾文本对齐的效果

可以看到它只对最后一行文本起作用。利用这些文字的对齐方式，可以解决表单对齐问题，如图 10.16 所示。



图 10.16 解决表单对齐问题的效果

使用 justify 属性后，效果如图 10.17 所示。

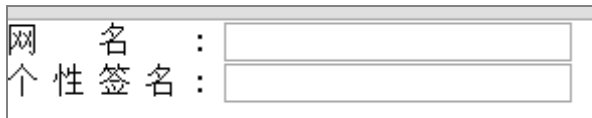


图 10.17 使用 justify 属性后的效果

图 10.17 所对应的代码如下。

```

<style>
  body,ul,li,label,input{
    margin:0;

```

```
padding:0;
}
ul{
list-style:none;
}
label{
width:100px;
display:inline-block;
text-align-last:justify;
}
</style>
<ul>
<li>
<label for="#user">网名:</label>
<input type="text">
</li>
<li>
<label for="#user">个性签名:</label>
<input type="underwrite">
</li>
</ul>
```

如果想让文字对齐，可以把“:”去掉，修改代码如下。

```
<style>
body,ul,li,label,input{
margin:0;
padding:0;
}
ul{
list-style:none;
}
li{
overflow:hidden;
}
label{
width:100px;
white-space:pre;
}
```



```

    label,input{
        float:left;
    }
</style>
<ul>
    <li>
        <label for="#user">网    名:</label>
        <input type="text">
    </li>
    <li>
        <label for="#user">个性签名:</label>
        <input type="underwrite">
    </li>
</ul>

```

效果如图 10.18 所示。

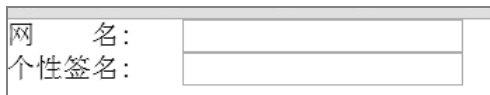


图 10.18 去掉“:”后的效果

图 10.18 所示效果的原理就是使用 `white-space` 保留空格。此段代码的缺点就是得自己调整空格，不过比起用“` `”实体效果要好得多。

`justify` 还可以用在如图 10.19 所示的情况下。

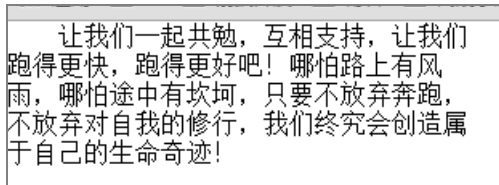


图 10.19 `justify` 可能会用到的地方

此段文字没有对齐，比较杂乱，看起来十分别扭，这时可以用 `text-align` 属性的 `justify` 值修改，代码如下。

```

<style>
    body,div{
        margin:0;
        padding:0;
    }

```

```

    }
    div{
        width:300px;
        text-align:justify;
        text-indent:2em;
    }
</style>

```

<div>让我们一起共勉，互相支持，让我们跑得更快，跑得更好吧！哪怕路上有风雨，哪怕途中有坎坷，只要不放弃奔跑，不放弃对自我的修行，我们终究会创造属于自己的生命奇迹！ </div>

效果如图 10.20 所示。

让我们一起共勉，互相支持，让我们跑得更快，跑得更好吧！哪怕路上有风雨，哪怕途中有坎坷，只要不放弃奔跑，不放弃对自我的修行，我们终究会创造属于自己的生命奇迹！

图 10.20 用了 justify 属性以后的效果

有时候只想让最后一行文本居中，如图 10.21 所示。

让我们一起共勉，互相支持，让我们跑得更快，跑得更好吧！哪怕路上有风雨，哪怕途中有坎坷，只要不放弃奔跑，不放弃对自我的修行，我们终究会创造属于自己的生命奇迹！

图 10.21 让最后一行文本居中的效果

想让最后一行文本居中时，代码如下。

```

<style>
    body,div{
        margin:0;
        padding:0;
    }
    div{
        width:300px;
        text-align:justify;
        text-align-last:center;
    }

```

```
</style>
<div>让我们一起共勉，互相支持，让我们跑得更快，跑得更好吧
！哪怕路上有风雨，哪怕途中有坎坷，只要不放弃奔跑，不放弃对
自我的修行，我们终究会创造属于自己的生命奇迹！</div>
```

需要注意的是，如果同时设置 `text-align` 和 `last-align-last`，那么会忽略 `text-align` 对最后一行的设置。

10.2.3 文本书写模式

除了以上的文字对齐属性，在 CSS 3 中还新增了一个文字书写模式：`writing-mode`。它有以下几个属性。

- `horizontal-tb`：水平方向从上到下的书写方式；
- `vertical-rl`：垂直方向从右到左的书写方式；
- `vertical-lr`：垂直方向从左到右的书写方式；
- `lr-tb`：水平方向从上到下的书写方式。效果和 `horizontal-tb` 一样（在 IE 浏览器中）；
- `tb-rl`：垂直方向从右到左的书写方式。效果和 `vertical-rl` 一样（在 IE 浏览器中）。

举个例子，代码如下。

```
<style>
  div{
    width:150px;
    height:100px;
    outline:1px solid #dedede;
  }
  div:nth-of-type(1){
    writing-mode:horizontal-tb;
  }
  div:nth-of-type(2){
    writing-mode:vertical-rl;
  }
  div:nth-of-type(3){
    writing-mode:vertical-lr;
  }
  div:nth-of-type(4){
    writing-mode:lr-tb;
  }
}
```

```
div:nth-of-type(5){
    writing-mode:tb-rl;
}
</style>
<div>水平方向从上到下的书写方式</div>
<div>垂直方向从右到左的书写方式。</div>
<div>垂直方向从左到右的书写方式。</div>
<div>水平方向从上到下的书写方式</div>
<div>垂直方向从右到左的书写方式。</div>
```

效果如图 10.22 所示。

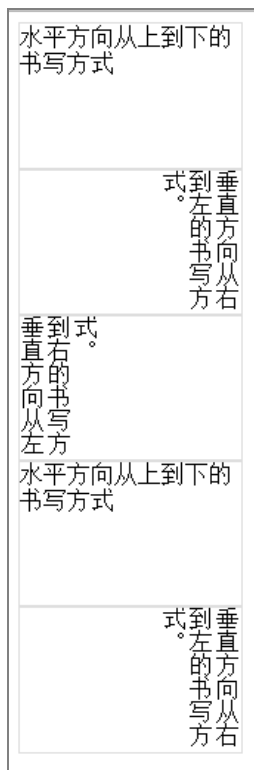


图 10.22 综合例子

有了这些属性，就可以对文本进行更多方式的排列了。

10.3 关于文字的一些其他属性

10.3.1 将超出宽度的文字隐藏

文字超出一定的范围就让它隐藏。举个例子，代码如下。

```
<style>
  div{
    width:150px;
    outline:1px solid #dedede;
  }
</style>
<div>明年的今天，你在干啥。</div>
```

未隐藏超出文字的效果，如图 10.23 所示。

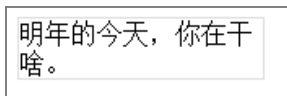


图 10.23 未隐藏超出文字的效果

现在只想让文字在一行显示，超出的文字显示为省略号，代码如下。

```
<style>
  div{
    width:150px;
    overflow:hidden;
    text-overflow:ellipsis;
    white-space:pre;
    outline:1px solid #dedede;
  }
</style>
```

效果如图 10.24 所示。

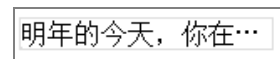


图 10.24 超出的文字显示为省略号

将超出的文字隐藏，用 `text-overflow` 设置超出的文字用省略号显示，还需要设置 `white-space` 让文字禁止换行。因为如果换行了文字就会在下一行显示，没有“超出”的效果了。部分浏览器需要设置 `width` 属性。

当然，也可以自己制作文字隐藏效果。自己制作文字隐藏效果的好处就是可以自定义隐藏时的效果，代码如下。

```
<style>
  .box{
    width:200px;
    height:18px;
    overflow:hidden;
  }
  .box .item{
    float:left;
    margin-right:23px;
  }
  .box:after{
    content:'...';
    float:right;
    margin-top:-38px;
  }
</style>
<div class="box">
  <div class="item">
    明年的今天，你在干啥。明年的今天，你在干啥。</div>
</div>
```

效果如图 10.25 所示。




图 10.25 自定义超出文字的隐藏效果

当文字内容不够时，这个伪元素会“跑上去”；而如果当内容够多的时候，伪元素会“占领”文字的位置。

除了用 CSS，还可以用 JavaScript 来实现超出文字隐藏的效果。就是用判断文字长度的 `length` 属性，如果 `length` 大小到达指定个数时就让它隐藏。

还可以用文字截取的方法，截取 $0 \sim n$ 个字，多出的不截取就可以了，代码如下。

```

<style>
  .box{
    width:200px;
    height:18px;
  }
</style>
<div class="box">明年的今天，你在干啥。明年的今天，你在干啥。</div>
<script>
  var oBox = document.querySelector('.box');
  oBox.innerHTML = oBox.innerHTML.substr(0,8) + '...';
</script>

```

效果如图 10.26 所示。



图 10.26 用 JavaScript 实现超出文字的隐藏效果

在实际操作的过程中，有时文字可能不是显示一行就隐藏了，而是超过两行才隐藏。在 webkit 内核的浏览器中有一个 `line-clamp` 属性，它可以用来限制一个块元素显示的文本行数，代码如下。

```

<style>
  div{
    width:150px;
    overflow:hidden;
    text-overflow:ellipsis;
    display:-webkit-box;
    -webkit-line-clamp:2;
    -webkit-box-orient:vertical;
    outline:1px solid #dedede;
  }
</style>
<div>明年的今天，你在干啥。明年的今天，你在干啥。</div>

```

效果如图 10.27 所示。

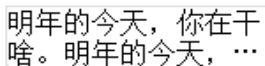


图 10.27 实现多行显示

需要注意的是，在使用 `line-clamp` 属性时，必须结合其他的 `webkit` 属性。常见结合的属性有以下两个。

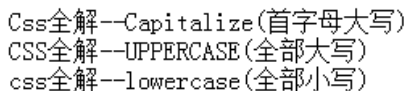
- `display: -webkit-box;`：必须结合的属性，将对象作为弹性伸缩盒子模型显示；
- `-webkit-box-orient`：必须结合的属性，设置或检索伸缩盒对象的子元素的排列方式。

10.3.2 字母转换大小写

CSS 还有一个将字母转换大小写的属性——`text-transform`。举个例子，代码如下。

```
<style>
  .box div:nth-of-type(1){
    text-transform:capitalize;
  }
  .box div:nth-of-type(2){
    text-transform:uppercase;
  }
  .box div:nth-of-type(3){
    text-transform:lowercase;
  }
</style>
<div class="box">
  <div>css 全解--capitalize(首字母大写)</div>
  <div>css 全解--uppercase(全部大写)</div>
  <div>css 全解--lowercase(全部小写)</div>
</div>
```

效果如图 10.28 所示。



Css全解--Capitalize(首字母大写)
CSS全解--UPPERCASE(全部大写)
css全解--lowercase(全部小写)

图 10.28 字母转换大小写

使用 `text-transform` 属性转换字母大小写就方便多了。

10.4 总结

本章所介绍的只是在 CSS 3 中常用的一些文字属性，感兴趣的读者可以学习一下 `word-break` 和 `word-wrap` 这两个属性，在实际操作中也会经常用到。

- 通过 `text-shadow` 属性可以制作文字阴影，但不对文字本身起作用。
- 通过 `text-fill-color` 属性可以设置文字填充色。
- 通过 `text-stroke` 属性可以对文字进行描边。
- CSS 3 中对 `text-align` 新增了属性，其中最常用的是 `justify` 属性，其作用是将文字两端对齐，不过它并不会对最后一行起作用。如果需要对最后一行也起作用，需要使用 `text-align-last` 属性。
- `text-align-last` 属性用来设置最后一行文本的对齐方式。
- 如果想改变文字的书写模式，可以使用 `writing-mode` 属性。
- `text-transform` 属性可以用来实现字母的大小写转换。

第 11 章 内容生成技术——用 CSS 来计数

在 CSS 中有两个很好用的伪元素，即“::before”和“::after”。也许你会看到别人会写成“:before”和“:after”，其实后面的这种写法是为了兼容早期的浏览器。万维网联盟（W3C）为了将伪类和伪元素区分，在伪元素前面加了两个冒号。伪元素有很多非常好用的属性，下面将详细介绍。

11.1 伪元素

- **::before**：用来将内容插入到元素的前面；
- **::after**：用来将内容插入到元素的后面。

::before 和 **::after** 中有一个 **content()** 属性用来生成内容，代码如下。

```
<style>
  .item1::before{
    content:'Hello';
  }
  .item1::after{
    content:'World';
  }
</style>
<div class="item1"> item1 </div>
```

效果如图 11.1 所示。

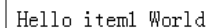
Hello item1 World

图 11.1 生成内容

需要注意的是，大部分浏览器将这两个伪元素默认以行内元素的方式显示，有了这两个伪元素，可以减少对标签的使用。

伪元素和标准元素的区别

看着好像伪元素和标准元素没什么区别，只不过是一个写在 CSS 中，一个写在 HTML 中。但是事实并非如此。

11.2 CSS 计数器

CSS 中有一个计数功能，就像使用变量一样，它有以下 4 个属性。

- **counter-reset**: 创建或重置计数器；
- **counter-increment**: 增长计数器；
- **content**: 生成内容；
- **counter()**: 将计数器的值添加到元素中。

使用 CSS 计数器，必须首先创建它，代码如下。

```
<style>
  ul{
    list-style:none;
    counter-reset:num;
  }
  ul>li::before{
    counter-increment:num;
    content:'List ' counter(num) ': ';
  }
</style>
<ul>
  <li>AAAAAA</li>
  <li>BBBBBB</li>
  <li>CCCCCC</li>
</ul>
```

效果如图 11.2 所示。

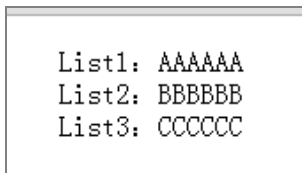


图 11.2 CSS 计数器

需要注意的是，必须使用伪元素来生产内容，不然不起效果。

不过它并不是按照元素的个数来计算值的，而是根据 `counter-increment` 的数量来计算的，但两个 `counter-increment` 的值必须一样，比如下面这个例子。

```
<style>
  ul{
    list-style:none;
    counter-reset:num;
  }
  ul>li::before,h2::before{
    counter-increment:num;
    content:'List' counter(num) ' ';
  }
</style>
<ul>
  <li>AAAAAA</li>
  <li>BBBBBB</li>
  <li>CCCCCC</li>
</ul>
<h2>DDDDDD</h2>
```

效果如图 11.3 所示。



图 11.3 根据 `counter-increment` 的数量计算

有了这个属性，如果需要这种效果就不需要去写 JavaScript 了。下面看一个复杂点的例子，代码如下。

```
<style>
  body{
    counter-reset:title;
  }
  h1::before{
    counter-increment:title;
    content:'The ' counter(title) ' ';
  }
</style>
<h1>Title</h1>
<h2>这是一段内容.</h2>
<h2>这是一段内容.</h2>
<h1>Title</h1>
<h2>这是一段内容.</h2>
<h2>这是一段内容.</h2>
<h1>Title</h1>
<h2>这是一段内容.</h2>
<h2>这是一段内容.</h2>
```

效果如图 11.4 所示。

The 1 Title
这是一段内容.
这是一段内容.

The 2 Title
这是一段内容.
这是一段内容.

The 3 Title
这是一段内容.
这是一段内容.

图 11.4 给 body 加 counter-reset

代码看上去好像没什么问题,但为什么要给 body 加 counter-reset 呢?其实是有原因的,counter-reset 表示创建或重置计数器。那么问题来了,在什么情况下,它表示创建呢?又在什么情况下它表示重置呢?

笔者试验几次之后所得出的结论是,当第一次使用 counter-reset 时,表示创建;当第二次使用时就表示重置。但要注意的是,是同一个元素才会重置。一个页面只有一个 body,所以对于只需要增加而不需要重置的计数器就可以写在 body 里。当然,也可以写在其他元素里,只要保证它是唯一的。举一个例子,代码如下。

```
<style>
  body{
    counter-reset:title;
  }
  h1::before{
    counter-increment:title;
    content:'The ' counter(title) ' ';
  }
  h1{
    counter-reset:content;
  }
  h2::before{
    counter-increment:content;
    content:'contetn ' counter(title) '.' counter(content
) ' ';
  }
</style>
<h1>Title</h1>
<h2>这是一段内容.</h2>
<h2>这是一段内容.</h2>
<h1>Title</h1>
<h2>这是一段内容.</h2>
<h2>这是一段内容.</h2>
<h1>Title</h1>
<h2>这是一段内容.</h2>
<h2>这是一段内容.</h2>
```

效果如图 11.5 所示。

The 1 Title

contetn 1.1 这是一段内容.

contetn 1.2 这是一段内容.

The 2 Title

contetn 2.1 这是一段内容.

contetn 2.2 这是一段内容.

The 3 Title

contetn 3.1 这是一段内容.

contetn 3.2 这是一段内容.

图 11.5 只要保证它是唯一的

利用计数器来实现获取 input checked 选择的数量。

代码如下。

```

<style>
  body{
    counter-reset: checked;
  }
  input:checked{
    counter-increment: checked;
  }
  div::after{
    content:counter(checked) '个分类';
  }
</style>
CSS3 <input type="checkbox">
HTML5 <input type="checkbox">
Javascript <input type="checkbox">
<div>你一共选择了</div>

```

效果如图 11.6 所示。

CSS3 ☒ HTML5 ☒ Javascript ☐
你一共选择了2个分类

图 11.6 利用 input checked 来计算选择的数量

其实可以自己手动更改 counter-increment 数值大小，如图 11.7 所示。

5 ☒ 2 ☒ 7 ☐ 1 ☒
Count 8

图 11.7 手动更改 counter-increment 数值大小

代码如下。

```
<style>
body{
    counter-reset:num;
}
input:nth-of-type(1):checked{
    counter-increment:num 5;
}
input:nth-of-type(2):checked{
    counter-increment:num 2;
}
input:nth-of-type(3):checked{
    counter-increment:num 7;
}
input:nth-of-type(4):checked{
    counter-increment:num 1;
}
div::after{
    content:'Count ' counter(num);
}
</style>
5<input type="checkbox">
2<input type="checkbox">
7<input type="checkbox">
```



```
1<input type="checkbox">
<div></div>
```

通过这段代码可以很好地实现图 11.8 的效果，如图 11.8 所示。counter 还支持传递一个值用来改变显示的序号，如图 11.9 所示。

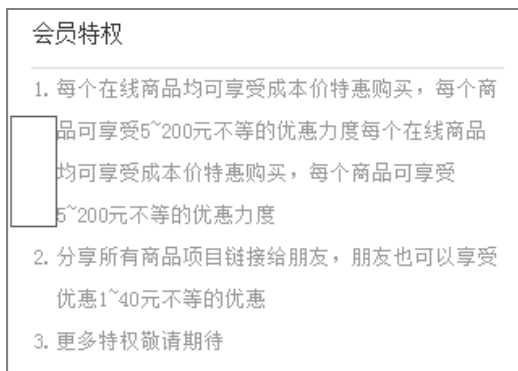


图 11.8 效果图

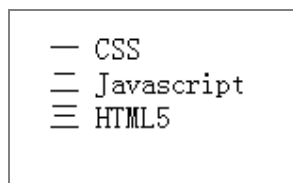


图 11.9 改变显示的序号

代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  ul{
    list-style:none;
    counter-reset:a;
  }
  li{
    counter-increment: a;
  }
  li::before{
    content:counter(a,cjk-ideographic) ' ';
  }
</style>
<ul>
  <li>CSS</li>
  <li>Javascript</li>
```

```
<li>HTML5</li>
</ul>
```

此段代码中用了“`cjk-ideographic`”，它会按照大小数字来显示。

那么 `cjk-ideographic` 的值是哪里来的呢？它就是“`list-style-type`”里面的值。`list-style-type` 属性还有其他的一些值，这里就不一一列举了。

11.3 content 的其他用途

在 `content` 中还有两个值。

- `attr`：插入标签属性值；
- `url`：插入一个外部资源（图像、音频、视频或浏览器支持的其他任何资源）。

使用 `attr` 的效果，如图 11.10 所示。



图 11.10 使用 `attr` 的效果

代码如下。

```
<style>
  .box a{
    position:relative;
  }
  .box a img{
    width:200px;
  }
  .box a::after{
    content:attr(title);
  }
```

```
        position: absolute;
        bottom: 0;
        left: 0;
        width: 100%;
        height: 35px;
        line-height: 35px;
        text-align: center;
        background-color: #ccc;
        color: #fff;
    }
</style>
<div class="box">
    <a href="javascript:;" title="这是一张风景图片">
        
    </a>
</div>
```

11.4 总结

伪元素的意义就在于它可以使 HTML 更加语义化。有些时候为了某些效果，不得不添加一些无意义的标签，而这两个伪元素既能起到这种辅助布局的作用，又不需要增加无意义的标签。

第 12 章 解决让人头疼的布局

在 CSS 中，布局可谓是“灵魂”所在，对于不同的网页，有不同的布局风格。本章主要介绍可自适应的布局。掌握可自适应的布局需要对浮动、定位等知识点有比较深的了解，可自适应的布局就是内容会随着屏幕的大小而变化，但不会影响原有的布局。

12.1 制作可自适应的布局

12.1.1 左侧固定、右侧自适应的布局

如图 12.1 所示，是一个左侧固定、右侧自适应的布局。



图 12.1 左侧固定、右侧自适应的布局

要实现这个效果，只需要把前一个元素设置为左浮动就可以了，后一个元素千万不要设置为浮动，不然第二个元素高度超过了第一个元素就会换行。代码如下。

```
<style>
body,div,h1{
margin:0;
padding:0;
}
```

```
.box > img{  
    width:100px;  
    height:100px;  
    float:left;  
}  
  
</style>  
  
<div class="box">  
      
    <div class="content">  
        <h1>内容内容内容内容内容内容内容内容内容内容内容  
内容内容内容内容内容内容内容内容内容内容内容内容内容  
内容内容内容内容内容内容内容内容内容</h1>  
    </div>  
</div>
```

给.content 设置左边距就可以了，效果如图 12.3 所示。

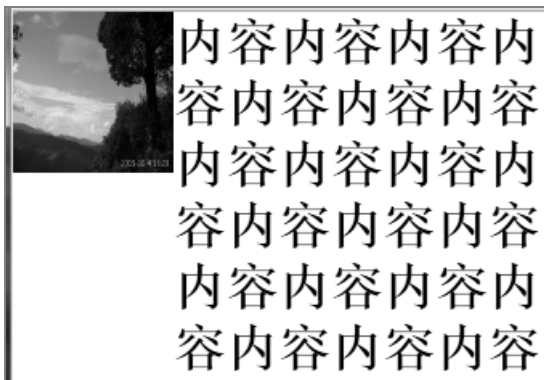


图 12.3 设置左边距

这样一个可自适应的布局就完成了，不过别忘了给.box 盒子增加 overflow 的 hidden 属性，避免父元素无法自适应浮动元素的高度。当然，最简单的办法就是创建一个 BFC。

12.1.2 右侧固定、左侧自适应的布局

如图 12.4 所示，是一个右侧固定、左侧自适应的布局。



图 12.4 右侧固定、左侧自适应的布局

原理和左侧固定、右侧自适应是一样的，只需要给第一个元素设置右浮动就可以了，代码如下。

```
<style>
body,div,p{
    margin:0;
    padding:0;
}
.box{
    height:35px;
    line-height:35px;
    overflow:hidden;
}
```

```
.box a{
    float:right;
    width:80px;
    height:100%;
    text-align:center;
    color:#fff;
    background-color:orange;
    text-decoration:none;
}
.box p{
    height:100%;
    text-align:center;
    background-color:#ccc;
}
</style>
<div class="box">
    <a href="javascript:;">确定</a>
    <p>价格: ¥ 120</p>
</div>
```

除了使用浮动，还可以使用定位来实现此效果，代码如下。

```
<style>
    body,div,p{
        margin:0;
        padding:0;
    }
    .box{
        position:relative;
        height:35px;
        line-height:35px;
        overflow:hidden;
    }
    .box a{
        position:absolute;
        right:0;
        top:0;
        width:80px;
        height:100%;
```

```

        text-align:center;
        color:#fff;
        background-color:orange;
        text-decoration:none;
    }
    .box p{
        height:100%;
        text-align:center;
        background-color:#ccc;
    }
</style>

```

效果如图 12.5 所示。

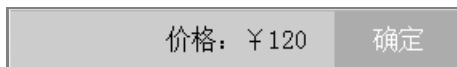


图 12.5 利用定位实现此效果

通过比较会发现，使用这两种方法实现相同的效果还是有差别的。使用定位做出来的效果，前面的文字明显没对齐，并且一旦屏幕宽度过小，文字一部分会被定位的元素盖住。导致出现这个问题的原因就是，定位的那个元素根本不占任何空间，所以它的居中自然是包括 a 标签的那段空间。在屏幕比较小的时候就“露馅了”，所以还是使用浮动比较好。

12.1.3 多列文字垂直居中

如图 12.6 所示，多列文字垂直居中的布局。



图 12.6 多列文字垂直居中的布局

代码如下。

```

<style>
    body,div,p{
        margin:0;
        padding:0;
    }

```



```
.box{
    border:1px solid #dedede;
    height:80px;
    line-height:80px;
}
</style>
<div class="box">
    <p>12 个月</p>
    <p>会员剩余有效期: 365 天</p>
</div>
```

将行高和高度设置成一样的，它们就会居中对齐，但对于多行这么操作显然是不行的，效果如图 12.7 所示。



图 12.7 多行情况下用 line-height 不行

实际上，如果使用 padding 属性就简单多了，代码如下。

```
.box{
    border:1px solid #dedede;
    padding:20px 0;
}
```

效果如图 12.8 所示。但是使用 padding 属性也有一个缺点，就是如果把其中一个元素去掉，你会发现它的高度变矮了，如图 12.9 所示。



图 12.8 使用 padding 属性的效果



图 12.9 使用 padding 属性的缺点

也就是说,如果想让它的高度是固定的,使用 `padding` 还是有问题的,不过这种方法偶尔也会用到。所以 `padding` 虽然简单,但不适合在大部分情况下使用。下面来看一个更好的方法,代码如下。

```
<style>
  body,div,p{
    margin:0;
    padding:0;
  }
  .box{
    display:table;
    width:100%;
    height:60px;
    border:1px solid #dedede;
  }
  .box .cell{
    display:table-cell;
    vertical-align:middle;
  }
</style>
<div class="box">
  <div class="cell">
    <p>12个月</p>
    <p>会员剩余有效期: 365 天</p>
  </div>
</div>
```

效果如图 12.10 所示。



12个月 会员剩余有效期: 365天

图 12.10 利用 `display:table` 的特性

利用 `display:table` 的特性就可以很好地解决这个问题了。用定位也可以实现此效果,代码如下。

```
<style>
  body,div,p{
```

```

margin:0;
padding:0;
}
.box{
position:relative;
height:60px;
border:1px solid #dedede;
}
.box > div{
position:absolute;
left:0;
top:0;
bottom:0;
height:36px;
margin-top:auto;
margin-bottom:auto;
}
</style>
<div class="box">
  <div>
    <p>12 个月</p>
    <p>会员剩余有效期: 365 天</p>
  </div>
</div>

```

效果如图 12.11 所示。



图 12.11 利用定位

这种效果的实现和垂直居中的效果差不多。用 CSS 转换（transform 函数）同样可以实现此效果，代码如下。

```

<style>
body,div,p{
margin:0;
padding:0;

```

```

    }
    .box{
        position:relative;
        height:60px;
        border:1px solid #dedede;
    }
    .box > div{
        position:absolute;
        top:50%;
        transform:translateY(-50%);
        -webkit-transform:translateY(-50%);
    }
</style>

```

效果如图 12.12 所示。



图 12.12 利用 transform

实现图 12.5 所示效果的原理就是先让所显示的文字距离上边 50%，然后通过 translateY 再让它下来 50%。你可能觉得这么做是矛盾的。其实定位 top 的 50% 是相对于 .box 的，而 translate 是相对于自己元素本身的。简单地说，就是先距离 .box 50%，然后再减去自己元素的一半高度，这样就居中了，这个与 “margin-top:-元素高度的一半” 是一样的，代码如下。

```

<style>
    body,div,p{
        margin:0;
        padding:0;
    }
    .box{
        position:relative;
        height:60px;
        border:1px solid #dedede
    }
    .box > div{
        position:absolute;

```

```

    top:50%;
    margin-top:-18px;
  }
</style>

```

这种方法和上面的效果一样，但缺点是必须得知道元素的高度。

12.2 利用伸缩盒模型进行布局

在 CSS 3 中新增了一种布局方式，即伸缩布局（Flex）。

如果用伸缩布局，那么前面介绍的布局就可相对简单地实现，代码如下。

```

.box{
    display:flex;
    align-items:center;
    justify-content:center;
    height:60px;
    border:1px solid #dedede;
}
<div class="box">
    <p>Flex</p>
</div>

```

效果如图 12.13 所示。

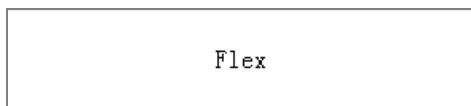


图 12.13 垂直水平居中的布局

使用伸缩布局实现左侧自适应、右侧固定宽度的效果，代码如下。

```

<style>
    body,div,p{
        margin:0;
        padding:0;
    }
    .box{
        display:flex;

```

```

        margin-top:100px;
        height:60px;
        line-height:60px;
        border:1px solid #dedede;
    }
    .box p{
        flex:1;
        height:100%;
        text-align:center;
        background-color:#ccc;
    }
    .box a{
        width:80px;
        height:100%;
        text-align:center;
        background-color:orange;
        color:#fff;
        text-decoration:none;
    }
</style>
<div class="box">
    <p>Flex</p>
    <a href="javascript:;">确定</a>
</div>

```

效果如图 12.14 所示。

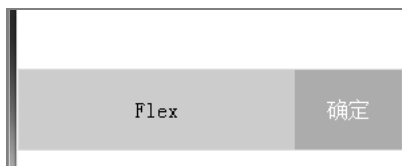


图 12.14 左侧自适应、右侧固定宽度的布局

如果需要使用伸缩布局，得先把 `display` 属性设置成 `flex` 或者 `inline-flex`。如果想兼容浏览器还得加浏览器前缀。但移动端不支持这个属性，其支持的还是老版本的 `display:-webkit-box;`；所以如果需要对移动端兼容，得写两套代码，不过现在有相关的库可以解决这个问题。从这两段代码中，可以发现使用伸缩布局比使用浮动布局更加简洁，接下来就让我们深入地学习一些关于伸缩布局的知识。

12.2.1 伸缩盒模型基础

下面针对伸缩盒模型举一些例子。

display 显示方式如下。

- flex: 块 flex;
- inline-flex: 行内 flex。

(1) 举个 flex 的例子，代码如下。

```
<style>
  body,div,p{
    margin:0;
    padding:0;
  }
  .box{
    display:flex;
    height:35px;
    line-height:35px;
  }
  .box p{
    background-color:pink;
  }
</style>
<div class="box">
  <p>Flex 1</p>
  <p>Flex 2</p>
</div>
```

效果如图 12.15 所示。

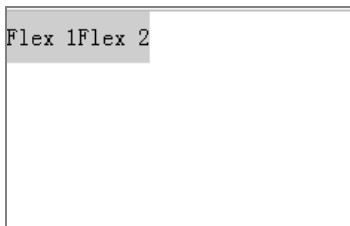


图 12.15 flex 的例子

效果类似 display:inline-block;。再举一个例子，代码如下。

```
<style>
  body,div,p{
    margin:0;
    padding:0;
  }
  .box{
    display:flex;
    width:500px;
    height:35px;
    line-height:35px;
  }
  .box p{
    width:100%;
    text-align:center;
  }
  .box p:nth-of-type(1){
    background-color:pink;
  }
  .box p:nth-of-type(2){
    background-color:green;
  }
</style>
```

效果如图 12.16 所示。



图 12.16 效果图

明明给 p 标签设置的是 100%，却没有超出父元素的宽度。先看下面这段代码。


```
<style>
  body,div,p{
    margin:0;
    padding:0;
  }
  .box{
    display:flex;
    width:500px;
    height:35px;
    line-height:35px;
    background-color:red;
  }
  .box p{
    width:30%;
    text-align:center;
  }
  .box p:nth-of-type(1){
    background-color:pink;
  }
  .box p:nth-of-type(2){
    background-color:green;
  }
</style>
```

效果如图 12.17 所示。

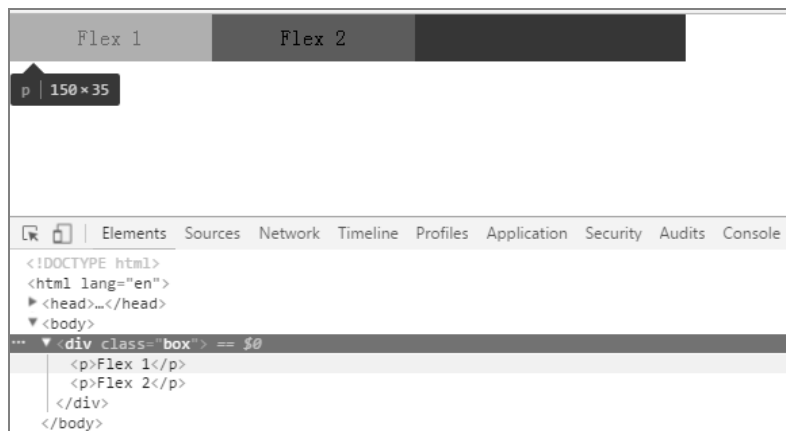


图 12.17 子元素的宽度超过了父元素

也就是说，如果子元素的宽度超过了父元素，超出的部分会被截断。

(2) 举个 `inline-flex` 的例子，代码如下。

```
<style>
  body,div,p{
    margin:0;
    padding:0;
  }
  .box{
    display:inline-flex;
    width:500px;
    height:35px;
    line-height:35px;
    background-color:red;
  }
  .box p{
    width:100%;
    text-align:center;
  }
  .box p:nth-of-type(1){
    background-color:pink;
  }
  .box p:nth-of-type(2){
    background-color:green;
  }
</style>
<div class="box">
  <p>Flex 1</p>
  <p>Flex 2</p>
</div>
```

效果如图 12.18 所示。



图 12.18 `inline-flex` 的例子

你会发现 `display:inline-flex` 简直和 `display:flex` 一模一样，而实际上它们还是有所区别的，代码如下。

```
<style>
  body,div,p{
    margin:0;
    padding:0;
  }
  div{
    height:35px;
    line-height:35px;
  }
  .item1{
    display:flex;
    background-color:orange;
  }
  .item2{
    display:inline-flex;
    background-color:pink;
  }
</style>
<div class="item1">flex</div>
<div class="item2">inline-flex</div>
```

效果如图 12.19 所示。

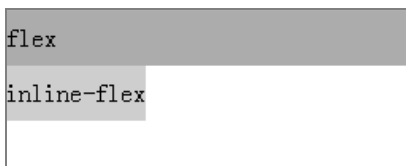


图 12.19 flex 和 inline-flex 的区别

你可以想象成 `flex` 是 `block`，而 `inlin-flex` 是 `inline-block`。也就是说 `flex` 有块级元素的特性，而 `inline-flex` 有 `inline-block` 的特性。如果还不能理解，看下面这段代码。

```
<style>
  body,div,p{
    margin:0;
    padding:0;
```

```

    }
    div{
        height:35px;
        line-height:35px;
    }
    .item1{
        display:flex;
        width:300px;
        background-color:orange;
    }
    .item2{
        display:inline-flex;
        background-color:pink;
    }
</style>
<div class="item1">flex</div>
<span>能和我一起站吗? </span>
<div class="item2">inline-flex</div>
<span>好基友</span>

```

效果如图 12.20 所示。

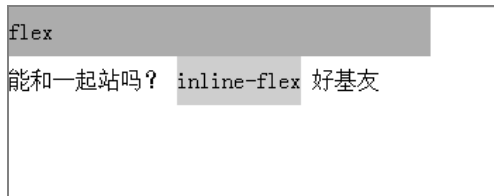


图 12.20 对比图

flex-direction 设置 flex 模型的方向。

- row: 从左到右，从顶部到底部（默认值）；
- row-reverse: 从右到左，1 在最右边，2 在从右到左的第 2 个位置，以此类推；
- column: 垂直排列；
- column-reverse: 垂直排列并且逆转。

justify-content 设置水平方向的对齐。

- flex-start: 左对齐（默认）；
- flex-end: 右对齐；

- center: 居中对齐;
- space-between: 两端对齐;
- space-around: 每个项目两侧的间隔相等。

align-items 设置垂直方向的对齐。

- stretch: 未设置高度或设为 auto, 将占满整个容器的高度 (默认);
- flex-start: 顶端对齐;
- flex-end: 底部对齐;
- center: 中间对齐;
- baseline: 被定位在容器的基线。

(3) 举一个 stretch 的例子。

你会发现应用了 stretch 的元素高度被拉伸了, 如图 12.21 所示。



图 12.21 应用了 stretch 的元素高度被拉伸

代码如下。

```
<style>
  body,div,p{
    margin:0;
    padding:0;
  }
  div{
    display:flex;
```

```

        width:500px;
        height:100px;
        outline:1px solid #dedede;
    }
    .item1{
        align-items:stretch;
    }
    .item2{
        align-items:flex-start;
    }
</style>
<div class="item1">
    <p>stretch</p>
</div>
<div class="item2">
    <p>flex-start</p>
</div>

```

(4) 举一个 Baseline 的例子，代码如下。

```

<style>
    body,div,p{
        margin:0;
        padding:0;
    }
    .item,.item1{
        display:flex;
        height:100px;
        background-color:pink;
    }
    .item p:nth-of-type(1),.item1 p:nth-of-type(1){
        font-size:30px;
    }
    .item{
        align-items:baseline;
    }
    .item1{
        align-items:flex-start;
    }
</style>

```

```

<div class="item">
  <p>这是</p>
  <p>baseline</p>
</div>
<div class="item1">
  <p>这是</p>
  <p>flex-start</p>
</div>

```

效果如图 12.22 所示。

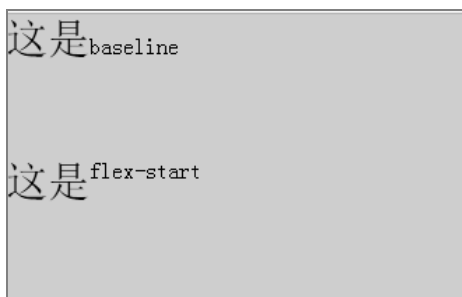


图 12.22 baseline 的例子

这就是 stretch、baseline 和其他几个值的区别。

flex-wrap 是否换行呢？

- nowrap: 不换行（默认值）；
- wrap: 换行；
- wrap-reverse: 以相反的顺序换行。

align-content 设置换行后的对齐方式。

- stretch: 高度平分，把剩余空间占满（默认值）；
- flex-start: 紧跟着上一个元素，都在开始处；
- flex-end: 紧跟着上一个元素，都在结束处；
- center: 垂直居中；
- space-between: 上下均匀地排列；
- space-around: 上下均匀地排列、环绕。

order 指定顺序。

- number: 谁的值大谁就在后面。

举一个例子，代码如下。

```

<style>
  body,div,p{
    margin:0;
    padding:0;
  }
  .box{
    display:flex;
  }
  .box p:nth-of-type(1){
    order:6;
  }
  .box p:nth-of-type(2){
    order:2;
  }
  .box p:nth-of-type(3){
    order:5;
  }
</style>
<div class="box">
  <p>CSS(1,6)</p>
  <p>HTML5(2,2)</p>
  <p>Javascript(3,5)</p>
</div>

```

效果如图 12.23 所示。

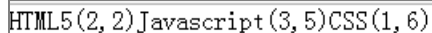


图 12.23 order 指定顺序

order 可以是负数。

align-self 给予元素单独设置垂直方向的对齐方式。

- flex-start: 被定位在容器的顶部;
- flex-end: 被定位在容器的底部;
- center: 被定位在容器的中心;
- baseline: 被定位在容器的基线;
- stretch: 项目被拉伸以适应父元素。

说明一下 `align-self` 属性的意思，如果想对某个元素单独设置对齐方式，那么可以使用这个属性，它将会覆盖 `align-items` 设置的值。

(5) 举个 `flex` 指定某个元素所拥有的份数的例子，代码如下。

```
<style>
  body,div,p{
    margin:0;
    padding:0;
  }
  .box{
    display: flex;
    width: 300px;
    margin-top: 12px;
  }
  .box div:nth-of-type(1){
    flex: 1;
    background-color: red;
  }
  .box div:nth-of-type(2){
    flex: 2;
    background-color: pink;
  }
  .box div:nth-of-type(3){
    flex: 3;
    background-color: orange;
  }
</style>
<div class="box">
  <div>1</div>
</div>
<div class="box">
  <div>1</div>
  <div>2</div>
</div>
<div class="box">
  <div>1</div>
  <div>2</div>
```

```
<div>3</div>
</div>
```

效果如图 12.24 所示。

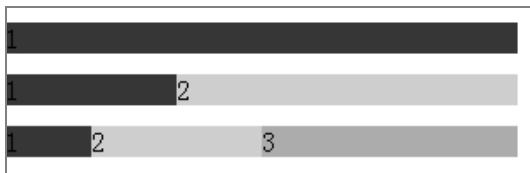


图 12.24 设置 flex 份数的例子

12.2.2 伸缩盒模型进阶

当只要一个子元素（只包含一级元素）的时候，flex 不管设置多少都是一样的，代码如下。

```
.box div:nth-of-type(1){
    flex:20;
    background-color:red;
}
```

效果如图 12.25 所示。



图 12.25 效果图

只要 flex 的份数一样就会平分，代码如下。

```
.box div:nth-of-type(1){
    flex:20;
    background-color:pink;
}
.box div:nth-of-type(2){
    flex:20;
    background-color:orange;
}
```

效果如图 12.26 所示。

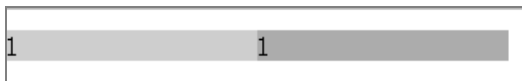


图 12.26 只要 flex 的份数一样就会平分

flex 的份数不能小于 1，代码如下。

```
.box div:nth-of-type(1){
    flex:-1;
    background-color:pink;
}
.box div:nth-of-type(2){
    flex:0;
    background-color:orange;
}
```

效果如图 12.27 所示。



图 12.27 flex 的份数不能小于 1

举个简写 flex-direction 和 flex-wrap 的例子，代码如下。

```
<style>
  body,div,p{
    margin:0;
    padding:0;
  }
  .box{
    display:flex;
    width:300px;
    margin-top:12px;
    flex-flow:row-reverse nowrap;
  }
</style>
<div class="box">
  <div>再坚持一会，马上完结。</div>
  <div>OKOKOKOKOKOKOKOKOKOKOKOKOKOK。</div>
</div>
```

效果如图 12.28 所示。

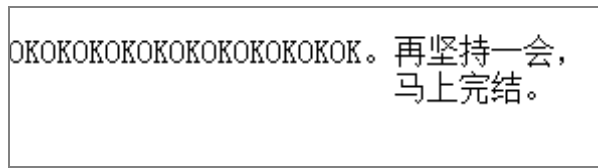


图 12.28 简写 flex-direction 和 flex-wrap 的例子

flex-flow 属性用法：flex-direction flex-wrap。

在不设置 flex 份数的情况下，子元素不会继承父元素的宽，这和 display:block;还是有区别的，代码如下。

```
<style>
  body,div,p{
    margin:0;
    padding:0;
  }
  .box{
    display:flex;
    width:200px;
    height:100px;
    margin-top:12px;
    background-color:pink;
  }
  .box p{
    background-color:orange;
  }
  .box + .box p{
    flex:1;
  }
</style>
<div class="box">
  <p>下一秒</p>
</div>
<div class="box">
  <p>下一秒</p>
</div>
```

效果如图 12.29 所示。

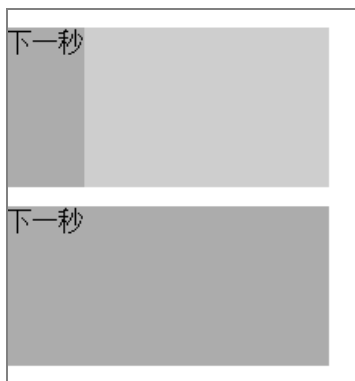


图 12.29 flex 和 block 的区别

12.2.3 伸缩盒模型实战

如果想得到图 12.30 所示的效果，一般都用浮动来写，但现在有了伸缩布局就简单多了，可以用伸缩布局来实现此效果，如图 12.30 所示。



图 12.30 利用伸缩布局来实现此效果

代码如下。

```
<style>
    .box{
        display: flex;
        width: 300px;
        height: 35px;
        line-height: 35px;
    }
    .box .item{
        flex: 1;
        border: 1px solid #ccc;
        text-align: center;
    }
    .box .item:not(:first-of-type){
        margin-left: -1px;
    }

```

```

</style>
<div class="box">
  <div class="item">css</div>
  <div class="item">html</div>
  <div class="item">javascript</div>
</div>

```

如果想使宽度自适应，只要去掉宽度就可以了，代码如下。

```

.box{
  display: flex;
  height: 35px;
  line-height: 35px;
}

```

效果如图 12.31 所示。

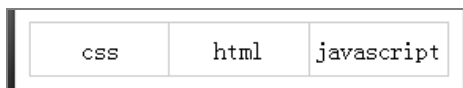


图 12.31 宽度自适应的效果

如果想在一定宽度范围里面自适应，代码如下。

```

.box{
  display: flex;
  min-width: 220px;
  max-width: 640px;
  height: 35px;
  line-height: 35px;
  margin: 0 auto;
}

```

效果如图 12.32 所示。



图 12.32 在一定宽度范围里面自适应的效果

举个多列垂直居中的例子，代码如下。

```

<style>
  .box{
    display: flex;

```

```

        width:200px;
        height:200px;
        flex-direction:column;
        justify-content:center;
        border:1px solid #dedede;
    }
    .box .item{
        height:35px;
        line-height:35px;
        border-bottom:1px solid #ccc;
        text-align:center;
    }
    .box .item:first-of-type{
        border-top:1px solid #ccc;
    }
}
</style>
<div class="box">
    <div class="item">css</div>
    <div class="item">html</div>
    <div class="item">javascript</div>
</div>

```

效果如图 12.33 所示。



图 12.33 多列垂直居中的例子

需要注意的是，这里千万不要使用 flex 份数。如果使用了 flex 份数，子元素高度会拉伸，就不能实现多列垂直居中的效果了。

如果将这段代码放在移动端的 UC 浏览器或微信中，就不能实现此效果了。主要原因是伸缩盒模型被修改过好几次名字，现在 UC 等浏览器还是用的之前老版本的 `display:box`。虽然 `display:box` 现在已经过时，但是 `flex` 在移动端的兼容性不好。不过有一个插件可以解决这个问题，即 `autoprefixer`。

第 13 章 飞越 CSS

13.1 CSS 最佳实践

一个块元素宽度默认继承包含块的宽度，因此有时宽度可省略。举个例子，代码如下。

```
<style>
  body,div{
    margin:0;
    padding:0;
  }
  .box{
    width:200px;
  }
  .item{
    height:35px;
    line-height:35px;
    background-color:orange;
  }
</style>
<div class="box">
  <div class="item">item</div>
</div>
```

效果如图 13.1 所示。



图 13.1 有时宽度可省略的效果

当边框颜色和文字颜色一样时可省略边框颜色。举个例子，代码如下。

```
<style>
    body,div{
        margin:0;
        padding:0;
    }
    .box{
        width:200px;
        height:35px;
        line-height:35px;
        text-align:center;
        color:red;
        border:1px solid;
    }
    .box:hover{
        color:orange;
    }
</style>
<div class="box">box</div>
```

默认情况下的效果如图 13.2 所示。当鼠标移动到这个 div 上时，如图 13.3 所示。

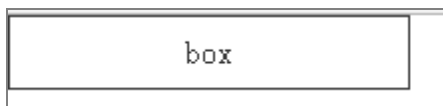


图 13.2 默认情况下的效果



图 13.3 hover 时的效果

出现图 13.2 和图 13.3 所示的效果，是因为在不设置边框颜色时，边框默认继承文字的颜色。

当文字颜色和其他属性（如 background、border）颜色一样时，可以使用 `currentColor` 关键字来继承它的颜色。

举个例子，代码如下。

```
<style>
    .box{
        width:100px;
        height:100px;
        color:red;
```

```

        background-color:currentColor;
        border:1px solid currentColor;
    }
    .box:hover{
        color:pink;
    }
</style>
<div class="box"></div>

```

效果如图 13.4 所示。



图 13.4 使用 currentColor 的效果

当父元素样式和子元素的某个样式相同时，可以通过 `inherit` 来继承父元素样式。举个例子，代码如下。

```

<style>
    .box{
        width:100px;
        height:100px;
        border:1px solid red;
    }
    .box .item{
        width:50%;
        height:50%;
        border:inherit;
    }
</style>
<div class="box">
    <div class="item"></div>
</div>

```

效果如图 13.5 所示。

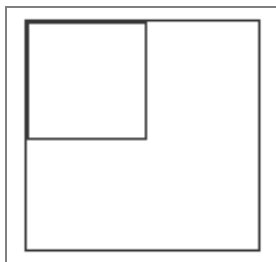


图 13.5 通过 `inherit` 来继承父元素样式的效果

如果想单独设置例如图 13.6 所示的效果，只想继承父元素的边框颜色，代码如下。

```
<style>
  .box{
    width:100px;
    height:100px;
    border:1px solid red;
  }
  .box .item{
    width:50%;
    height:50%;
    border:inherit;
    border-width:2px;
  }
</style>
```

需要注意的是，继承的写在前面，修改的属性放后面，防止被覆盖。

在某些情况下，父元素高度不是必须有的。举个例子，代码如下。

```
<style>
  .box{
    width:100px;
    border:1px solid red;
  }
  .box .item{
    height:100px;
  }
</style>
<div class="box">
  <div class="item"></div>
</div>
```

效果如图 13.6 所示。

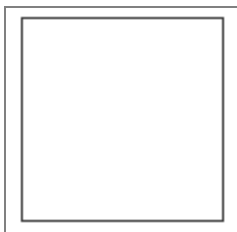


图 13.6 在某些情况下父元素的高度不是必须有的

在默认情况下，包含块会自适应其元素高度。但如果其元素使用了浮动，则需要给父元素添加 “overflow:hidden;” 或使用其他方法。

有时可以利用特殊符号替换小图标。举个例子，代码如下。

```
<style>
  .box{
    font-size:33px;
    color:red;
  }
</style>
<div class="box">👑</div>
```

效果如图 13.7 所示。



图 13.7 利用特殊符号替换小图标

13.2 纯 CSS 的世界

在深入了解 CSS 以后，你会发现很多以前需要用 JavaScript 来完成的效果，现在用简单的 CSS 代码就可以完成了。

13.2.1 利用 checked 选择器实现 tab 切换

利用 checked 实现 tab 切换的效果，如图 13.8 所示。

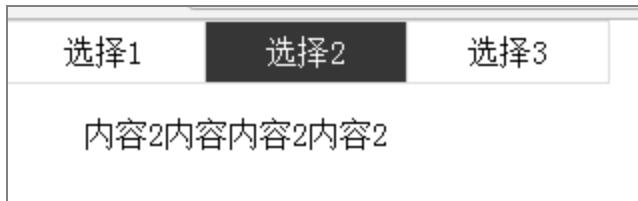


图 13.8 利用 checked 实现 tab 切换

代码如下。

```
<style>
  body,div,input,label{
    margin:0;
    padding:0;
  }
  #tab>input{
    opacity:0;
    position:absolute;
    left:0;
    top:0;
  }
  #tab .label{
    overflow:hidden;
  }
  #tab .label>label{
    float:left;
    width:100px;
    height:30px;
    line-height:30px;
    border:1px solid #dedede;
    text-align:center;
    margin-left:-1px;
  }
  .content li{
    display:none;
```

```

    }

    input:nth-of-type(1):checked~.label>label:nth-of-type(1){
        background-color:red;
        color:#fff;
    }
    input:nth-of-type(2):checked~.label>label:nth-of-type(2){
        background-color:red;
        color:#fff;
    }
    input:nth-of-type(3):checked~.label>label:nth-of-type(3){
        background-color:red;
        color:#fff;
    }
    input:nth-of-type(1):checked~.content li:nth-of-type(1){
        display:block;
    }
    input:nth-of-type(2):checked~.content li:nth-of-type(2){
        display:block;
    }
    input:nth-of-type(3):checked~.content li:nth-of-type(3){
        display:block;
    }
</style>
<div id="tab">
    <input checked type="radio" name="name" id="name1">
    <input type="radio" name="name" id="name2">
    <input type="radio" name="name" id="name3">
    <div class="label">
        <label for="name1">选择 1</label>
        <label for="name2">选择 2</label>
        <label for="name3">选择 3</label>
    </div>
    <div class="content">
        <ul>
            <li>内容 1 内容内容 1 内容 1</li>
            <li>内容 2 内容内容 2 内容 2</li>
            <li>内容 3 内容内容 3 内容 3</li>

```

```

    </ul>
  </div>
</div>

```

以上代码产生效果的原理就是单击 label 时会选中 input 标签，然后通过 CSS 让当前选中的那个元素的对应内容显示就可以了。

13.2.2 利用:target 选择器实现遮罩层效果

单击按钮时会弹出遮罩层，如果再单击遮罩层就将其隐藏了，如图 13.9 所示。

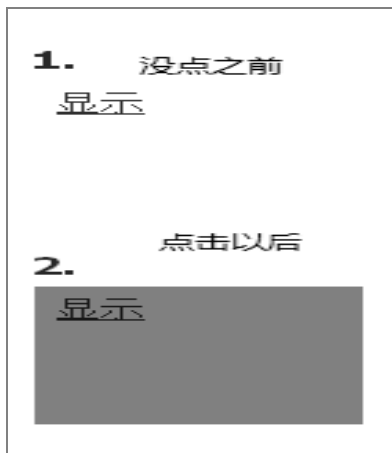


图 13.9 利用:target 选择器实现遮罩层效果

代码如下。

```

<style>
  #show{
    position:fixed;
    left:0;
    top:0;
    width:100%;
    height:100%;
    background-color:rgba(0,0,0,0.5);
    display:none;
  }
  a[href="#hide"]{
    position:absolute;

```



```

        left:0;
        top:0;
        width:100%;
        height:100%;
    }
    #hide:target{
        display:none;
    }
    #show:target{
        display:block;
    }
</style>

<a href="#show">显示</a>
<div id="hide">
    <div id="show">
        <a href="#hide"></a>
    </div>
</div>

```

13.2.3 scaleY 配合 animation 制作 loading

scaleY 配合 animation 制作 loading 的效果，如图 13.10 所示。



图 13.10 制作 loading

代码如下。

```

#box{
    position:relative;
}
#box>i{
    position:absolute;
    top:0;

```

```

        width:4px;
        height:35px;
        background-color:#0bac84;
        border-radius:5px;
    }
    #box>i:nth-of-type(1){
        left:0px;
        animation:loading 1s ease-in .1s infinite;
    }
    #box>i:nth-of-type(2){
        left:8px;
        animation:loading 1s ease-in .3s infinite;
    }
    #box>i:nth-of-type(3){
        left:16px;
        animation:loading 1s ease-in .6s infinite;
    }
    #box>i:nth-of-type(4){
        left:24px;
        animation:loading 1s ease-in .3s infinite;
    }
    @keyframes loading{
        0%{
            transform:scaleY(1);
        }
        50%{
            transform:scaleY(0.5);
        }
        100%{
            transform:scaleY(1);
        }
    }
    <div id="box">
        <i></i>
        <i></i>
        <i></i>
        <i></i>
    </div>

```

13.2.4 利用 hover 实现手风琴效果

利用 hover 实现手风琴效果，当鼠标没有移动到色块上时的效果如图 13.11 所示。



图 13.11 当鼠标没有移动到色块上时的效果

当鼠标移动到某个色块上时的效果如图 13.12 所示。



图 13.12 当鼠标移动到某个色块上时的效果

代码如下。

```
<style>
  body,div{
    margin:0;
  }
  .box{
    height:100px;
  }
  .item{
    float:left;
    width:25%;
    height:100%;
    transition:all 0.5s;
  }
  .item:nth-of-type(1){
    background-color:red;
  }
}
```

```

.item:nth-of-type(2){
    background-color:pink;
}
.item:nth-of-type(3){
    background-color:green;
}
.item:nth-of-type(4){
    background-color:orange;
}
.box:hover div{
    width:calc((100% - 40%) / 3);
}
.box .item:hover{
    width:40%;
}
</style>
<div class="box">
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
</div>

```

在此段代码中，核心的代码是最后两行。

13.2.5 利用 checked 选择器制作星星评分

单击五角星前的效果如图 13.13 所示。



图 13.13 单击五角星前的效果

单击五角星后的效果如图 13.14 所示。



图 13.14 单击五角星后的效果

以上代码产生效果的原理就是默认让全部的 `span` 显示为选中后的样式，然后通过 `checked` 选择器，让当前选择器后的所有 `span` 设置为未选中时的样式。但当前后一个 `span` 是需要选中的，用“兄弟”选择器即可。这里还用了一点小技巧，就是将 `input` 设置为绝对定位，但并没有设置 `left` 和 `top` 值，这样这些 `input` 还会在原来的位置上定位。只需要将这些 `input` 的透明度设置为 0 就可以了。

13.2.6 使用 flex 伸缩盒模型实现瀑布流布局

使用 flex 伸缩盒模型实现瀑布流布局的效果，如图 13.15 所示。



图 13.15 使用 flex 伸缩盒模型实现瀑布流布局的效果

代码如下。

```
<style>
  body,div{
    padding:0;
    margin:0;
  }
  .box{
    display: flex;
    flex-wrap: wrap;
  }
  .box img{
```

```

        flex-grow:1;
        margin:5px;
        height:200px;
        object-fit: cover;
    }
    @media (min-width:1200px){
        .box img:nth-last-of-type(-n + 5){
            flex-grow:0;
        }
    }
</style>
<div class="box">
    
    
    
    
    .....
</div>

```

在此段代码中给图片设置了 `flex-grow`，这个属性用来扩展图片的宽度，从而达到自适应包含块。如果不设置这个属性，则效果如图 13.16 所示。



图 13.16 在不设置 `flex-grow` 的情况下的效果

如图 13.16 所示，可以看到有一部分空白，这时设置 `flex-grow` 属性即可，但是设置 `flex-grow` 属性后图片进行了拉伸。因此给图片使用 `object-fit` 属性，这个属性用来防止图片被拉伸，它的使用方法和 `background-size` 非常像。

代码的最后使用了 CSS 媒体查询，主要是因为如果最后一行只有较少的图片时，可能会引起图片过渡变形，如图 13.17 所示。



图 13.17 在图片较少时可能出现的情况

如图 13.17 所示，可以看到最后一张图片严重变形了。因此如果最后一行图片较少，就不应该进行拉伸了。如果想更精确地控制图片的大小，还需要修改图片对应的 CSS 媒体查询的代码。

13.3 结束语

要想更深入地了解 CSS，不仅需要系统地学习理论知识，还需要亲自动手不断地实践探索。在编程的路上，希望大家更好地学习基础知识，只有基础巩固了，才能跟上编程的“潮流”。